A Simulation of the European Data Grid

Paul Crosby

October 31, 2002

Contents

List of Figures

List of Tables

1. Introduction

This report is intended to describe my work in developing a simulation of the running of computational jobs in the European Data Grid. The EDG itself has been developed collaboratively in the EU to harness unused computational power at its member sites, and analyse high energy physics data more efficiently. This will become increasingly important for the new generation of HEP experiments, as the enormous volumes of raw data to be processed will be too large to be dealt with effectively using conventional systems.

The EDG project is changing rapidly, as its structure and its software are modified by new ideas and trial and error. The various components that make up this large and complex structure are being developed by several independent teams, who have to coordinate regularly to make sure that the interfaces between these components remains consistent.

These interfaces, and the communication and flow of information between the components, that this work has attempted to simulate. A job can be generated, and then its lifetime can be followed as it is scheduled, submitted, queued, run and completed. The EDG components running at the simulated sites pass information to each other, coordinate the transfer of large data files needed to run a job, and distribute the jobs effectively between available machines.

This simulation has been used to carry out experiments involving different procedures for scheduling incoming jobs. The virtual jobs generated by the simulation require a number of large data files, which may be located in storage media at various different physically separated nodes belonging to the Grid, as well as a certain period of computer processing time to run over this data.

The scheduling algorithms, of varying degrees of complexity, are used by the Resource Broker to decide to which of the member Compute Elements a job, submitted from a User Interface, should be assigned to. It can use dynamic feedback about these sites, provided by the Grid's Information Services. The data must also transferred to the Storage Element local to the chosen Compute Element, and any changes in the distribution of data across the Grid must be registered with the Replica Catalog.

Different algorithms can affect the time taken to run jobs, both in terms of total throughput, or the performance of a single job. They can also optimise the use of the resources available, ensuring that the load of jobs is spread fairly and evenly across the Grid, or minimising the amount of network traffic. Sometimes this can be done in a complementary manner, while in other cases optimising performance from one perspective degrades it from another. This simulation is not intended to entirely replicate the functionality of each of the EDG software components, but instead to show how they interrelate, and how adjusting certain parameters can affect the Grid as a whole.

Section 2 discusses the concept of a Grid, and how the concept is emerging in analogy with the appearance of electrical power grids. Different types of Grid for applications with different requirements are discussed. Some of the operating systems and tools being developed to meet these needs are briefly described.

Section 3 outlines the nature and computing requirements of high energy physics, and how these requirements are growing rapidly with the new generation of large scale experiments. The main initiatives developing HEP Grids in the world are mentioned, and some possibilities for dealing with the computation problem are described. The concept of Regional Centres for HEP computing is also discussed.

Section 4 describes the European Data Grid and its structure as it stands. A brief description is given of the software components, i.e. the User Interface, Resource Broker, Replica Catalog, Information Services, Compute Element, Storage Element, and the Logging and Bookkeeping services (not included in this simulation).

Section 5 describes the benefits of a simulation of the EDG. Some of the available tools for Grid simulation are mentioned, and why they were passed over in favour of Ptolemy II, a less specific modelling tool.

Section 6 describes the mechanics of PTII, the principles of discrete event modelling and the construction of a PTII model. The simulated versions of the components described in section 4 are detailed, as well as the simplified network model used for the simulation. Various limitations of this simulation are then discussed.

Section 7 describes the two broad categories that scheduling algorithms fall into. The six algorithms tested in this work are then explained, based upon estimated traversal time, location of required data, minimum queue length, data location and minimum queue length in combination, random selection, and round-robin selection.

Section 8 describes five metrics to measure the performance of the Grid, in terms of completion time of the last-completed job, number of data file transfers, machine loading average, queue length, and job traversal time.

Section 9 discusses the first set of experiments, carried out in a homogenous Grid with ten identical member sites. The first experiment tests the six scheduling algorithms described in section 7 with a moderate load of jobs with similar CPU time requirements. In the second, the job is increased. The third experiment again uses a moderate job load, but using jobs whose CPU requirements are Gaussian distributed.

Section 10 uses the same setup as the first experiment described in section 9, but only using the round-robin scheduler. Jobs are generated for five different periods of time at the beginning of runs of the simulation, to check that the simulation behaves convincingly with different loads.

Section 11 replaces the homogenous Grid with a heterogenous one, including one site with very good computing and storage resources, three medium sized ones and six smaller ones, to mimic the Regional Centre structure. Two experiments are run using this Grid, with equally sized jobs, and jobs with Gaussian distributed sizes.

In Section 12, conclusions are drawn about the relative performances of the six scheduling algorithms, and how effectively the simulation has represented the behaviour of and EDG setup.

2. What Is A Grid?

A Grid is an infrastructure which allows computing resources which are distributed geographically, and may belong to many separate organisations, to be managed and utilised efficiently, and shared between the members of the Grid. It should run in such a way that it is 'invisible' to the user who wishes to harness the resources available, and will utilise new kinds of software to make the infrastructure for this resource sharing possible.

The name Grid is taken from the analogous development of national power grids in the last century. When electricity first became seen as a source of power for new kinds of machinery, light sources and so on, local generators were created to power the first light bulbs and other luxuries for people with access to these generators. However, demand began to grow for these new commodities, and eventually a national electricity network formed, able to aggregate the power generated across a nation, and supply it where needed. Lulls and surges in this demand at different times of day or year could be compensated for, providing a consistent supply of electricity for everyone, at a much reduced cost.

According to this analogy, we are currently at the stage of localised production, and the tools to build Grids of computing power are rapidly emerging. The development of the electrical power grid had a huge impact on society, allowing great leaps in technology and quality of life. Another analogy to the Grid which is often cited is that of the appearance of national train and road networks, which allowed the growth of much bigger centres of population. Writers such as Foster and Kesselman(sp?) [1], often seen as the visionaries of the Grid, have claimed that the sharing of computing resources in such a way will have an impact upon society of a similar magnitude.

The Grid will not quite be a "next generation" version of the Web, despite some similarities. The World Wide Web, built on the HTTP protocol and the HTML language, allows users to share information around the world. A Grid goes much further than that, in that computing resources themselves are being shared. If you host a website, you are making the information it contains available for anyone to browse (barring security issues). Membership of a Grid is more complex than that.

The anonymity possible for internet users is not viable in a Grid where those users are being allowed access to your machine. All users will have to be authenticated for this trust to be established. This will enable their software to locate an appropriate resource for them, execute their task, and return the outcome to them without the user needing to know how or where it was carried out.

How precisely this will be achieved is a matter of ongoing research, and technology is emerging as many different potential users work to build the tools that they will need to make the Grid work for them. There are several classes of application that will benefit greatly from the power that a Grid will make available to them, and in some cases it will be possible to run applications which would have been impossible with the computing power available at any one site.

2.1. Grid Applications

The categories of Grid application described below are not completely distinct, as there is a good deal of overlap in some cases. They are also not mutually exclusive in principle, as there is no reason why a given machine or local site should not be able to run several types of application. However it is natural, at least at first, for separate Grids to be developed for different applications, as we learn how to make use of these distributed resources for these differing purposes. There are also security issues, as you are not just sharing information, as is the case with the World Wide Web, but access to your computing resources themselves.

Thus one may not be able to talk about 'The Grid' the way we talk about 'The Web', or at least not yet. But eventually it may become possible to create a unified grid with middleware able to deal with requests to run many different types of application, and map them onto the resources available to it, perhaps charging the user for the privilege of using someone else's machines, and crediting the owner of those machines. But for now we must concern ourselves with separate Grids, as we learn the techniques that will make this resource sharing possible.

The following sections will describe five categories of application: distributed supercomputing; high-throughput computing; on-demand computing; collaborative computing; and finally data-intensive computing. The 'Data Grid' with which this work is concerned falls into this last category.

2.1.1. Distributed Supercomputing

These applications require a large amount of processing power to solve problems that cannot be dealt with on the machines available at a given site. The resources of several sites can be harnessed, and the complex interrelated processes making up the application can be mapped onto the available processors.

This is done in such a way as to minimise the cost of communication between the processes, using a technique called directed acyclic graphs. This communication is very important, as the smooth and consistent running of the application depends upon it. Thus network quality of service is very important, so that the users of the application can rely on the application running as it should.

Examples of this include: virtual reality applications for the military, simulating a combat situation to train fighter pilots; simulations of complex processes in chemistry [2]; and simulations of the evolution of very large systems in cosmology [3].

2.1.2. High-Throughput Computing

High throughput applications require a large amount of processing power, in order to deal with a load of many independent, or only loosely coupled jobs. This can be done much more efficiently if a job allocation system can look for unused processor cycles, and take advantage of them.

A successful example is Condor [14], which matches jobs to resources based on ClassAds (from an analogy with 'classified ads'). These are lists of specifications submitted by the user describing the type of resource required, and by the provider of the resource, describing the type of jobs allowed to run on their resource. Condor makes the best use of processors by matching jobs to the best resources for them, acting opportunistically to take advantage of idle machines. It also manages the jobs as they run, resubmitting or reallocating them if necessary because of system failures.

Condor was originally developed for local sites, but it has been interfaced with the Globus toolkit [12], which is designed to manage resources in distributed locations, to create Condor G. A Condor job ClassAd is submitted to Condor G, which uses Globus to submit and monitor the job at a remote site.

2.1.3. On-Demand Computing

Grids will also be useful for users with short term requirements that cannot be handled locally. These requirements might be for computational power, software, data, or any combination of the three. In this case they will be 'hiring' the resources supplied to them by the Grid. The cost to the user will be determined by the quality of the resources supplied (or, conversely, the quality of the resources obtained may be determined by how much they can offer for the required services).

Examples include the NEOS [4] and NetSolve [5] systems. These allow the user to make requests for the use of resources located at remote sites. Similar technology has been used to process images from a scanning tunnelling microscope in real time, allowing much greater accuracy in controlling the STM's movement [8].

2.1.4. Collaborative Computing

Collaborative computing is similar to distributed supercomputing, except that the collective computing power at the users' distributed sites is used to create a shared virtual space. This could potentially be a great leap in human interaction between remote sites.

The simplest scenario is the next step on from a video conference, in which a virtual meeting room can be created, in which representations of those present can interact with each other. But they would also be able to take advantage of the space in their virtual meeting place, and present three dimensional representations of data, or create complex models in 3D.

The CAVE5D system [6] allows collaborative exploration of large geophysical datasets in this manner. NICE [7] is a virtual world for education and entertainment of children.

2.1.5. Data-Intensive Computing

The final category are the so called 'Data Grids'. The users of these applications require access to large volumes of data stored at many geographically distributed sites. This data needs to be transferred reliably and efficiently to an appropriate location for the user to process it in the desired way. This means that these applications are usually communication and computationally intensive as well.

This kind of system is being used by the astronomical community for the Digital Sky Survey [9], and also to process large quantities of meteorological data to aid with weather prediction [10]. Most importantly for the purposes of this report, high energy physics collaborations are planning to build data grids to manage the enormous volumes of data that will be generated by the next generation of particle physics experiments, and to make them available for analysis.

2.2. Tools for Grid Building

Whatever the type of application that Grids are being constructed for, there are properties that many will have in common. This is leading to the emergence of software tools known as middleware to perform these common tasks.

The Legion software [11] provides an object based approach to constructing a Grid. In this scheme, every entity in the Grid is represented by an object, and the Grid is thus put together by assembling the desired structure from these objects. This is useful for the distributed supercomputing approach.

The Globus organisation [12] has constructed a toolkit which provides a means of coordinating the remote running of jobs. It deals with security and authentication issues via X.509 certificates [13] held by the users, an implementation of the public key system. The user can make queries as to which resources are available, submit their jobs, query the job's status as it runs, and then retrieve the results once they have completed. A sophisticated information sharing system, passing updates on the status of resources to the scheduling system, aids in the effective allocation of resources to jobs.

Software to deal with resource allocation has also been created. The Condor software, and its Gridified relative Condor G were described earlier. Another such allocation and management tool is Nimrod [15], which also started life as a resource management tool for a local system, and it too has been adapted for use in a distributed network.

These systems are often complementary, and it can often be useful to take some or all of several software suites in order to construct the Grid which best suits one's purposes. The European Data Grid, which this report is concerned with, contains portions of Globus to act as its information services, and it uses Condor to assign resources to jobs. In fact the user submits his or her job using the Condor G interface to Globus.

3. A Grid for High Energy Physics

It is the nature of high energy physics experiments that they will produce many orders of magnitude more raw data than genuinely useful information. Every interesting event generated by interactions between particles is made difficult to see, as it is buried in large numbers of more common and less interesting ones.

The uninteresting data can be cut down significantly at run time by only triggering data acquisition on interesting looking events in your detector, but a large quantity of data will still be left at the end of a run of an experiment, and this needs analysis off line to extract the interesting physics. At present, a typical experiment has the raw data stored centrally in tape archives, from which data can be staged to disk when required. When a certain data set is requested by an institute, possibly in an entirely different part of the world, the data is transferred to that remote site and the analysis is run locally. The institutes collaborating on the experiment operate more or less independently, so if a second institute in the same city required the same data set, it is quite likely that they would transfer it over again from the original site, as they were unaware of a replica of that dataset in a physically much closer location.

This data is in the form of files whose size are often of the order of magnitude of a gigabyte. Many of these files will be generated by a large scale particle physics experiment - for instance, the CDF experiment at Fermilab in Illinois produces of the order of a petabyte in a year. Not only must this data be stored, but it must be made available to sites distributed around the world, and processing power must be available in order to analyse it. As well as this, large Monte Carlo jobs generate simulated data for testing purposes, a task which is even more I/O intensive.

The next generation of HEP experiments, such as those based at the Large Hadron Collider at CERN, are going to produce much more data, as they operate at higher collision rates, with bigger detectors with more readout channels. However, computing resources have not necessarily grown at the same rate, and there is a danger of them failing to cope with the demand.

This situation falls into the category of data-intensive computing, as described in the previous section, and inevitably the idea of a particle physics Grid or Grids has been touted as the solution. Several projects have been set up, both in Europe and in the US, to develop Grid technology for the HEP community. Many are using software such as Globus to do this, and if the task of creating a single unified Grid to serve particle physicists across the world seems a daunting one at the moment, the first steps have been taken.

In the US, the Particle Physics Data Grid (PPDG) [16] has been set up by the National Science Foundation to serve American HEP experiments. The GriPhyN project [17] has been similarly created by the Department of Energy. An initiative called iVDGL [18] aims to unify the US Grid networks.

The Large Hadron Collider experiments are working on their own Grid, called LCG [19]. A project called DataTAG [20] is concentrating more on the networking needs for Grids set up to analyse scientific data. The European Data Grid, which will be described in detail in the next section, is being created to serve the needs of the astronomy and biomedical communities as well as that of high energy physics. Again, these groups are working with each other to share technology and evolve together, as well as working with their US counterparts.

There are certain properties that will be shared by any Grid set up for the analysis of HEP data, and arguably many types of scientific data analysis. The Grid must be able to accommodate large volumes of data as it is generated, and make it available to those who need access to it. The work to be done in processing this data, the individual jobs, will be large in number, but mostly independent of each other. That is, one job may depend on the results of a previous job, but the problem of complex programs composed of many processes all needing to communicate with each other over the wide area network will not apply here. This means that the HEP Grid scenario is similar in many ways to the high-throughput computing situation, as well as the data-intensive one.

A system of data management must be created to keep track of all of the data in a HEP Grid, as not only must the data be registered when it is first made part of the Grid, but all of the copies, or replicas, of that data must also be registered, and identified with the original data from which it was copied. This description of data (or data about data) is called metadata, and a repository for this metadata will be needed, to answer queries about where a certain data set can be found, and most easily obtained from.

The resources available to users of the Grid can also be described using metadata, describing the specifications of storage and processors, as well as their current status in terms of available cache space or idle machines. This metadata, as well as that for the data, needs to be supplied effectively to those who need to use it. Therefore an information service is required to do this, and to keep the information as recent as possible, so that decisions are not being made based on old news.

Finally, and possibly most importantly, it must be decided how all of this information will be acted upon, and how decisions will be made on where submitted jobs will be sent. Should all users be allowed to allocate their own jobs, in an "every man for themselves" struggle for the best resources? Should there be a single resource scheduling entity that deals with all scheduling decisions? Or should the solution be somewhere in between, perhaps with a scheduler per institute, or per country? All of these possibilities will have their advantages and disadvantages, so it is up to those creating the Grid to make the best use of the available technology.

Consideration must also be given to the reality of how all of these resources will be coordinated. Users of the Grid may wish for experimental data to be as well distributed as widely as possible, but it will all start out at one point, for a given experiment. Thus the MONARC group working on computing for the LHC have devised the idea of Regional Centres [21]. The member sites of the Grid are categorised into Tiers, depending on their ability to supply data to the user.

There is one Tier-0 site. This is CERN, where the data is acquired from the experiment, and initially stored. There are significant computing resources here, so CERN also functions as a Tier-1 site. Tier-1 Regional Centres serve one or more nations, and are able to provide all of the required data capacity, services and support, as well as computation power. Tier-2 sites serve a nation or a region, with correspondingly lower grade resources. At the bottom of the hierarchy are individual site servers (Tier-3), and desktop machines (Tier-4).

In a sense this organisation of resources is separate to the development of Grid technology. It is more of an acknowledgement of the work that must be done to coordinate these resources, and make them available to become part of the Grid. Whatever form the applications and middleware that are eventually used will take, they will have to be mapped onto the real machines and storage facilities provided and used by the members of a HEP Grid.

4. The European DataGrid

The European Data Grid [22] is an EU funded initiative to create a Grid of resources for computational and data-intensive scientific work. It has been created to aid not just the analysis of data from high energy physics experiments, but also that produced in biomedical research, and the European Space Agency's Earth observation project. It is hoped that the EDG will be able to demonstrate the cross-discipline potential of Grid technology.

The EDG project is led by CERN, but with five other main partners (the European Space Agency, CNRS in France, INFN in Italy, NIKHEF in the Netherlands, and PPARC in the UK), and fifteen other associated partners from the Czech Republic, Finland, France, Germany, Hungary, Italy, the Netherlands, Spain, Sweden and the UK.

The not inconsiderable task of constructing the EDG is divided into twelve Work Packages, which make up four Working Groups: Testbed and Infrastructure; Applications; Computational and Grid Middleware; and Management and Dissemination.

The EDG structure is based on software components installed where appropriate at the member sites. Computing resources, which can be anything from a single machine to a farm of PCs, or larger multiprocessor machines, are managed by a Compute Element. The storage space on disk and tape available to for Grid use, containing permanently hosted files as well as temporary replicas, is managed by a Storage Element. The data on the SEs is registered with a Replica Catalog, which can supply information relating logical file names to the physical file names given to that data at various sites. Information about the CEs and SEs is regularly updated at the Resource Broker, which uses that information, along with information about data file replicas, to pass a job, submitted using the User Interface software, to the Job Submission Service.

The following sections will describe how how these components work, and how they interact with each other. A schematic diagram of these components and interactions is shown in fig. 1.

4.1. User Interface

The User Interface is the software used by the end user to submit his job to the Grid. This user must write a JDL (Job Description Language) file, which is in the form of a ClassAd, like those supplied by Grid resources. This will include the details of the binary to be run, any input data files required, and instructions on how to deal with output, if any. It also contains the requirements of the job for the machine it will run on, and the user's preferences as to the properties of these machines. This file is submitted using the Condor G interface to the Grid, and this information, the binary, and any local configuration files are passed on to the Resource Broker.

Fig. 2 shows the life cycle of a job.

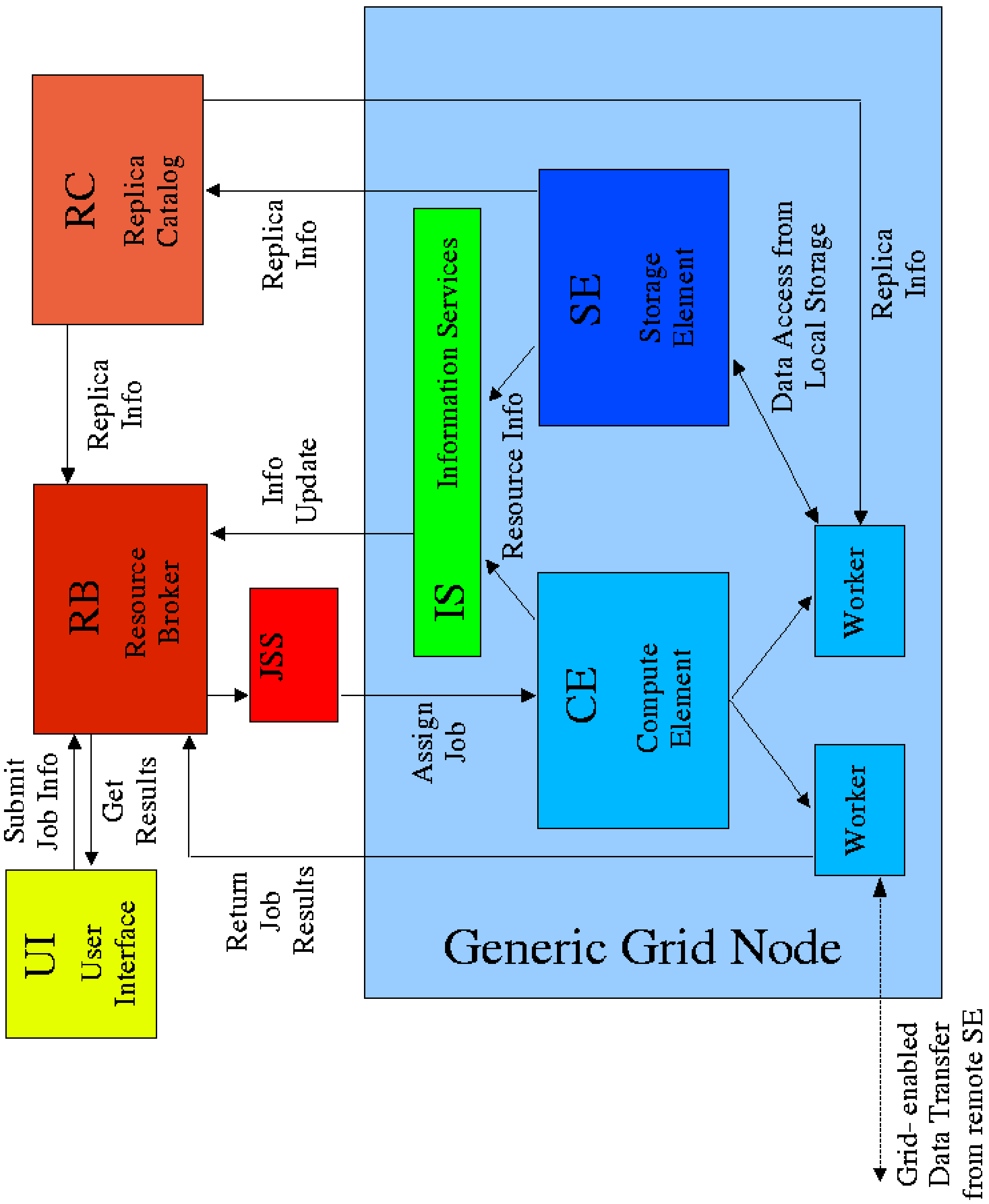

Figure 1: Relationships of EDG entities

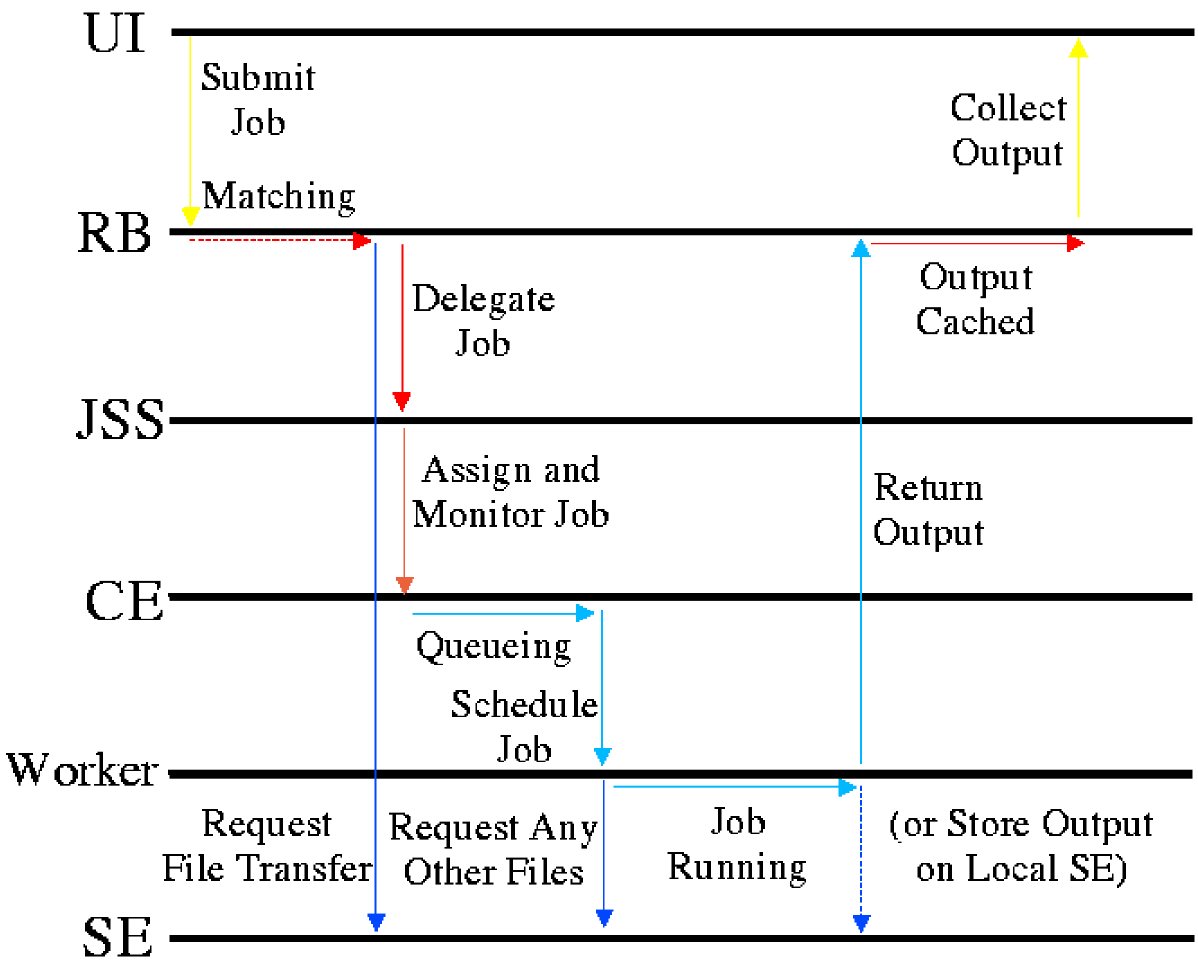

Figure 2: Stages in a job's lifetime, from submission through assignment and processing, to the return of the job's results

4.2. Resource Broker / Job Submission Service

All of the information being passed around is eventually used by the Resource Broker. The information sent to it by resources, and the user's JDL file, are in the form of ClassAds. These are compared by the Condor software, which attempts to find the most suitable resource to run the job on by matching the most suitable jobs to resources.

ClassAds contain a list of their attributes in the form:

```
Property = ‘‘Value’’
```

Followed by a description of the requirements and preferences for the properties of the ClassAd they are to be matched to, in the form:

```
Requirements = (other.Property1 > 5.0) && (other.Property2 == 1)
```

These ClassAd statements always evaluate to a value, in the case of the one above a boolean. However a quantity prefixed with "other" requires information from the ClassAd that this one is being matched with. If the statement evaluates to true in the context of the other ClassAd, then the two can be matched, otherwise it is not a valid combination.

In the job / resource matching case, the user will have supplied a list of the job's requirements, such as machine architecture and operating system. If the attributes of the system do not meet these requirements, that resource will be dropped from the list of candidates for the job. Similarly, the resource will supply a list of requirements of the jobs to be run on it, for instance specifying the maximum memory usage allowed.

The second phase of the matching is the ranking of ClassAds in order of preference. The user supplies a list of preferred attributes in a way similar to the Requirements statement, except that it evaluates to a number:

```
Rank = other.Property3 + (5 * other.Property4)
```

Evaluating the Rank statement in the context of the candidate resources will produce a different value for each, allowing the candidates to be put in order of preference. The job will be sent to the resource that is (in the user's eyes) the most favourable for that job.

The resource ClassAds are brought to the RB by the Information Services, updated when the previous information's lifetime has expired. However, much this information is changing dynamically, as jobs begin or end running on a resource. The RB can use the buffered information in its Information Index for the Requirements matching phase, as this tends to be based on attributes that do not change much. However for the Rank phase, it needs the most up to date information it can get, so it contacts the information services of the candidate resources directly for an update.

After a list of candidates has been produced in the Requirements phase, the RB will also contact the Replica Catalog if the job has supplied a list of input files that it needs, in the form of Logical File Names or Physical File Names at a particular site. The RC will resolve any LFNs into PFNs, and also supply information about the speed at which the required data can be moved (if necessary) to the candidate resources' local SE. The RB can then use this information to aid in resource selection if desired, and when a resource is selected, the best SEs from which to get the data will be passed on with the job in the resource-broker-info file.

Once a job has been scheduled to run on a resource, it is passed to the Job Submission Service. The JSS handles the actual submission of the job to its chosen resource, and once it is there it is responsible for monitoring that job on behalf of the user. If for any reason the job is no longer able to run on that resource, the JSS can retrieve it at return it to the RB for rescheduling. If the resource fails and the job is lost, the JSS can resubmit it when the resource is functioning again.

When the job has completed, the output is returned to the RB and placed in a cache for the user to collect, unless the user has specified an SE for the output to be stored on.

The sequence of these events is shown in fig. 3.

4.3. Replica Catalog

Every data file on the Grid must be registered with the Replica Catalog. It is given a logical filename (LFN) representing the file itself, and physical filenames (PFNs) represent an instance of that file on a particular SE.

Currently the RC only acts as a repository of this information, and when requested by the Resource Broker can resolve LFNs passed to it in terms of PFNs, and also pass the RB other information about the SEs to allow it to determine where best to get copies of a given set of files from. However, a future release of the RC will include a Replica Manager. This will learn from the patterns of file transfers as the Grid is running, and will preemptively move files between SEs in an attempt to anticipate demand.

4.4. Information Services

The structure of the EDG's Information Services is still being finalised, but its purpose is to make sure that information about resources (CEs and SEs) is made available as needed. Access to up to date information is important for effective scheduling of jobs.

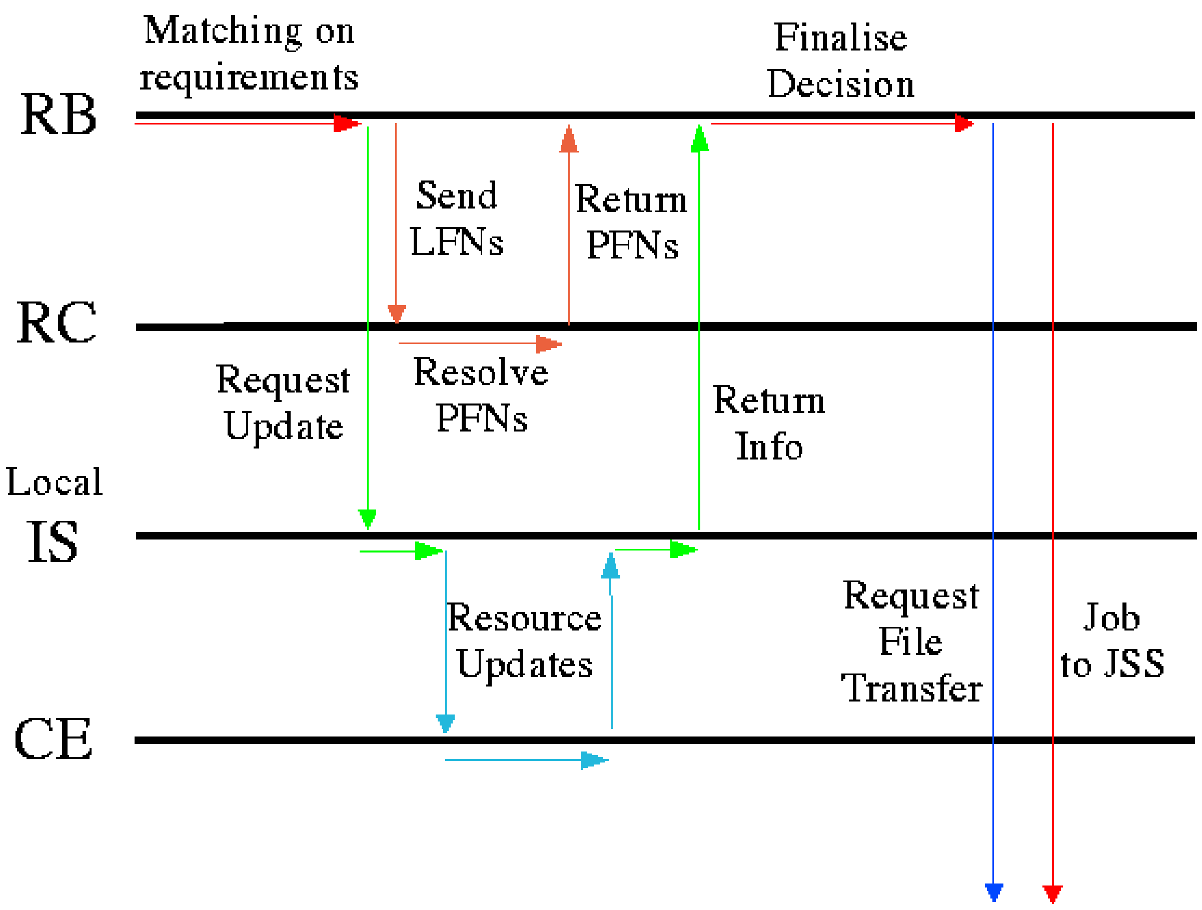

Figure 3: Sequence of interactions of the Resource Broker with the Replica Catalog and local Information Services, and decisions made

The resource information is published and made available by software at that resource called a Producer. This information is passed to any registered Consumers, which could be a repository of resource metadata or the Resource Broker itself. A Consumer can collect and summarise information from multiple Producers.

Consumers can also act as Producers, and pass the information that they have gathered to another Consumer which is collating data on an even larger scale, e.g. at the national or experiment level. Eventually the information gets to the Resource Broker which stores it in its Information Index, acting as the ultimate Consumer.

A Producer's information has a certain lifetime attached to it, and when it expires, new and up to date information is made available to Consumers. A Consumer can also query a Producer to obtain the most current data if required. In this way, after the Resource Broker has made initial decisions based on (largely) static characteristics of resources, it needs to make further choices based on more dynamic attributes, and thus needs to be sure it is acting upon information that is as current as possible. It will query the resources in question in its role as a Consumer, and they as Producers will supply the appropriate data.

4.5. Compute Element

The Compute Element acts as an interface and gatekeeper between the Grid at large, and a local system. It determines who should be allowed the use of the resources it governs, and what type of jobs should be allowed to run there. These user requirements and preferences are passed to the Resource Broker along with the details of the resource in the ClassAd.

Once a job has been accepted, it is passed to whatever queueing system is used at the site. However it is still monitored by the Job Submission service at this point, and when the job has finished running, its results will be returned to the Resource Broker to be collected by the user.

The CE also knows what its closest Storage Element is, and will supply this information to the Resource Broker so that it can arrange for any required input files to be copied to the local SE for a job to run. Any significant output from the job's running can also be stored at this SE until its owner collects it.

4.6. Storage Element

The Storage Element controls disk and tape space available to Grid applications, and provides information about the data stored in it. Again, it acts as an interface, as an application requesting a file does not care how physically that file is stored, only that it gets the file. Almost any work likely to be carried out on a CE will either require sizable input data files to run over, or (in the case of big Monte Carlo runs) disk space for the output to be written to. Thus it would be feasible to have an SE only at a given site, but impractical to set up a CE with no local SE.

An SE can have permanently hosted files, and cache space for temporary replicas of files hosted permanently elsewhere. Descriptions of this data, known as metadata, is passed to the Replica Catalog.

When a worker machine at a local CE needs a file, it contacts the SE, which will acquire a replica of the required file and place it in the cache.

4.7. Logging and Bookkeeping

The Logging and Bookkeeping services are just as important as any of the components mentioned above. They record every event in the lifetime of a job, to facilitate monitoring and troubleshooting in the Grid. However it has no real bearing on the running of the Grid, as it has a passive role. Therefore this project will not cover them, as there is no need for a simulation of the EDG to include these services.

5. Simulating the European Data Grid

It is clear that a Grid setup would be very beneficial for any organisation needing to get the maximum usage possible out of its computing resources. However, exactly how to do this is not quite so clear. A great many factors influence how well a system as large and complex as a Grid, and it is a difficult task even to determine how best to measure such a system.

A software simulation could save valuable time in determining the optimal parameters and configuration for a Grid, answering important questions without resorting to trial and error with a real testbed. By the nature of such a system, it can only be tested with a network of well (even globally) distributed sites with a realistically heavy workload to manage, and this obviously takes a great deal of manpower, cooperation, and time. If this network can be simulated to a sufficient degree of complexity, some of this effort can be saved by testing ideas out on the simulation instead.

For instance, the ClassAd matching method for job scheduling described earlier is a very flexible one, as users are free to select any resource attributes as their preference and thus affect the scheduling, and different users can be choosing different attributes. The choice of the best attributes to rank resources on is not a simple one to make, and will affect not just the running of the submitted job, but also that of the system as a whole. A simulation provides an opportunity to test different resource selection policies, and measure the effect that they have on system and job performance.

The information services could also benefit here, as a variety of different structures and hierarchies are possible, and will affect the system performance. A change in the configuration might make it easier for the RB to acquire the information it requires about resources, but at the cost of the currentness of the information. This will in turn affect how well the RB can distribute the workload across the Grid. These different structures can be tested much more easily if it is just a matter of changes to configuration of a simulation, rather than a major alteration to software installations across a large distributed network.

There are several Grid simulation tools in existence, but none of them were quite right for this project. Some were well developed but too specific, such as the MONARC project designed for simulation of distributed computing for Large Hadron Collider experiments. Others such as the GridSim toolkit, were not yet sufficiently developed at the beginning of this project. Eventually Ptolemy II was chosen, a design and modelling tool written in Java, which was not created specifically with Grid simulation in mind.

This might seem like an awkward and long way round to write this simulation, but every Grid is run in an appropriate way for its given purpose, and will be structured differently. There is more to be gained in terms of control of the way the simulation works than will be lost.

6. Grid Simulation with Ptolemy II

Ptolemy II can run in several different modes, but the interesting one for this project is "Discrete Event" modelling. Simulating the movement of jobs around a distributed network, and the information and data files also being moved around to facilitate this, is a complex process. A set of tasks which take a certain period to carry out, often concurrently with each other, must be kept track of, and coordinated within the flow of simulation time. The DE method is to see all of this as a series of discrete events representing the beginning and end of these tasks. The simulation is activated when one of these events occurs, and carries out the appropriate actions, including scheduling further events in the future. The simulation sleeps between these events, thus avoiding wasting time running when nothing is happening during a certain period of simulation time.

It should be pointed out that this is only 'sleeping' in the sense of simulation time. In real time the simulation jumps to the next event in the event queue and then activates the appropriate part of the simulation.

6.1. The Building Blocks of PTII

In PTII actions are carried out by Java objects called Actors, which can either be abstract entities to do arithmetic, or generate signals, and so on, or they can represent more solid entities such as the Actors designed for the scheduling simulation described later. These Actors communicate by exchanging Tokens, which are wrapper objects containing information that can be as simple as a double precision number, or a Java object in its own right. A simple schematic of an Actor is shown in fig. 4.

Any time an Actor carries out an action, or sends or receives a Token, it is considered an event. The events are placed in chronological order in the event queue, and then processed one by one in order. Thus the simulation is only running at these event times, and is inactive at all other times. Many events will result in another event at some point in the future; this future time is calculated, and an event is placed in the event queue, requesting that the appropriate Actor or Actors are woken up at that time to carry out the necessary action. Events cannot be removed from the queue, but if a queued event is rendered obsolete by other preceding events, the Actor can determine that it has been unnecessarily activated, and remain dormant.

The Actors are connected together in the GUI (called Vergil). Each Actor has one or more ports, which can send Tokens, or receive them, or both. Restrictions can be placed on the type of Token that the port can handle. Ports can be set up as single ports, connected to one other Actor, or as multiports, able to connect to many Actors.

An example of a Ptolemy II application is shown in fig. 5 (this is in fact the simulation described later). The PoissonClock, TimedDelay, UserInterface and Backbone

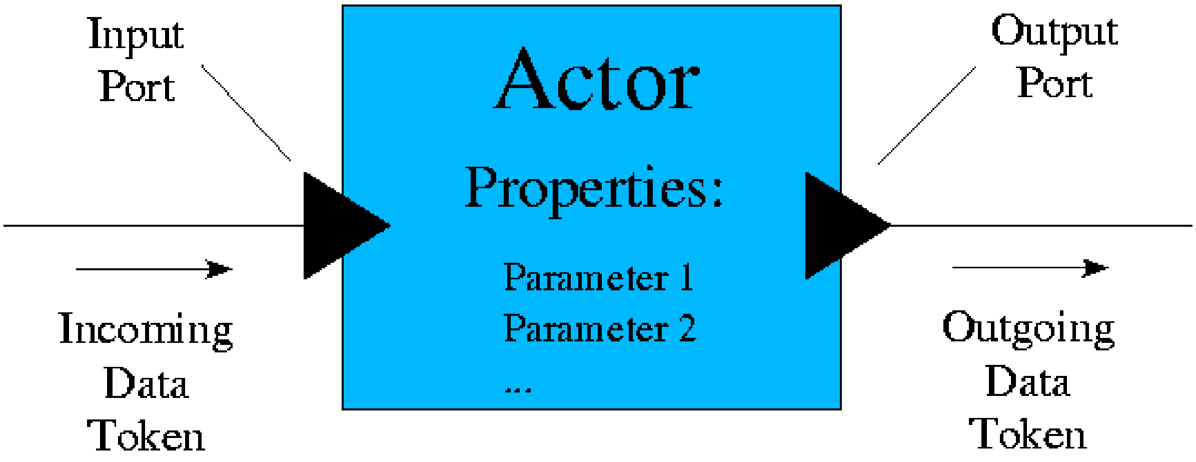

Figure 4: Schematic representation of a generic Actor object. In principle an Actor can have many parameters, and ports allowing data tokens in, out or both.

objects are Actors. The green DE Director controls various global characteristics of the application. The objects labelled Site 0, Site 1 etc. are composite Actors.

In fig. 6 we can see the contents of one of these composite Actors. We can see a variety of different Actor objects, as well as a port object on the left which represents outward connectivity to the rest of the simulation objects.

The Ptolemy ethos is that any Actor should be as general as possible in its applicability, ideally being 'domain polymorphic', i.e. usable in domains other than Discrete Event, and interconnectable with other Actors in the package. It was decided that this was not an important principle for this work, and could be abandoned. The object-oriented structure of Ptolemy II is the more important aspect, but with the objects involved heavily customised. Instead of exchanging strings, doubles etc, the Actors in this simulation would have to exchange more complex objects representing the interactions between the EDG entities.

6.2. Customised Actors

There is a large selection of Actors supplied with Ptolemy II, some specifically for the discrete event domain, but most designed to work in all of PTII's running modes. The supplied Actors are simple in function on the whole, intended to perform a small simple task in combination with many other Actors, and communicating by passing simple messages to each other as mentioned above.

This simulation is intended to represent the interaction of the various entities in an EDG setup. PTII presents two possibilities for this representation: composite Actors,

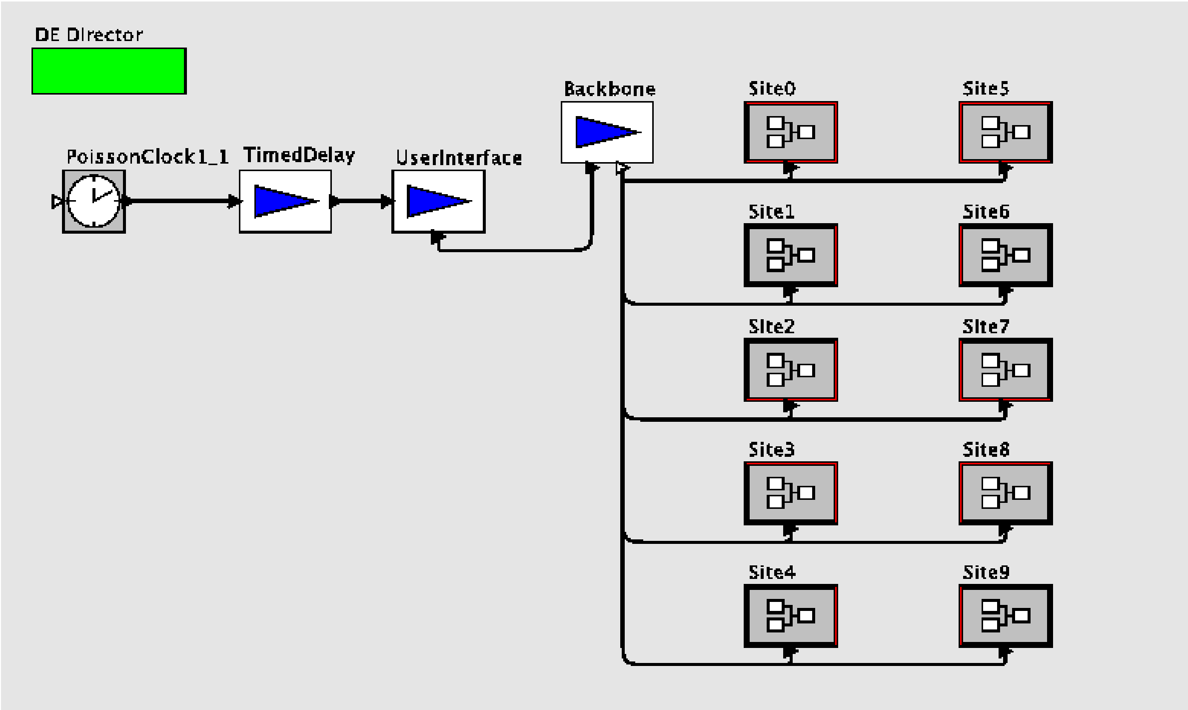

Figure 5: Screen grab of the layout of a Ptolemy II application.

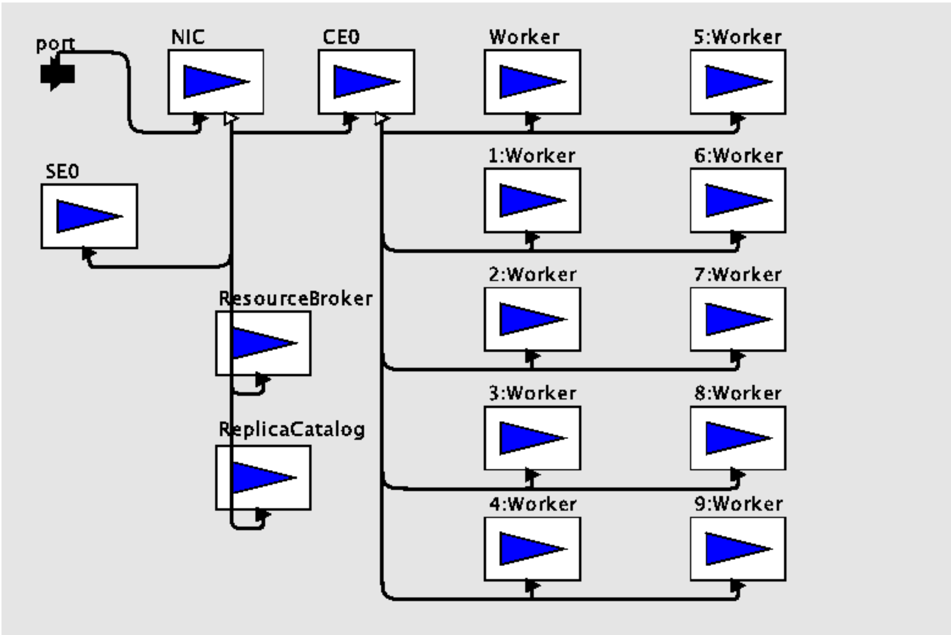

Figure 6: Screen grab of the contents of a composite Actor.

which are made up of the simplistic prewitten Actors in a specific configuration (with these composites then being combined), or new customised Actors.

The customised Actor route would save time in writing new code, but configuring them would be a sizeable task in itself. Reconfiguring them for a new experiment with the simulation might also involve repeating much of this work. Passing a message between these composite Actors would also be difficult, as the problem with simple message types would still persist.

Writing new customised Actors solves these problems, as a single Actor representing an entity would be easier to configure, and they can be designed to pass objects of the desired complexity. In this case even a simple number being passed would also have to have information describing its destination attached, a little like a URL to guide the passing of this message to the correct place. For these reasons, it was decided that new Actors would be written.

6.2.1. A Simplified Network Model

A realistic network simulation is itself a complicated task to carry out, and could take a significant proportion of the time available to write a Grid simulation. Since completely accurate network behaviour is not the most important factor for this work, it was decided that a simplified model should be devised, which could act as a placeholder, and be replaced with a proper network simulation later if so desired.

A "spider" shaped structure for the simulation was decided upon. This involves a central Backbone Actor, representing the network. Connected to this would be all of the sites making up the Grid, with each site containing CEs and SEs, and where appropriate a Resource Broker or Replica Catalog. Fig. 7(a) shows this simplified topology. 7(b) shows a more realistic topology for comparison, with some structure in the connection between sites.

The Backbone itself is not an intelligent entity, but merely knows about all of the sites attached to it, and when it receives a message, job, etc, it can read the desired destination and pass it on to the correct place. The intelligence in these communications lies in a NIC Actor, so called because it acts a little like a network interface card, and handles the interaction of the components at that site with the outside world.

Smaller messages, such as updates being passed through the information services, can be dealt with simply, and sent on to their destination as if they were traversing the network more or less instantaneously. However, data files can be of the order of a gigabyte in size, and will take some time to transfer to their destination even in a simplified network. The time taken for such a transfer is handled by the NICs at either end. A data file transfer is handled as follows.

An initial message is sent by the SE which has the file in question to its destination. This message includes the size of the file to be transferred. When the message passes

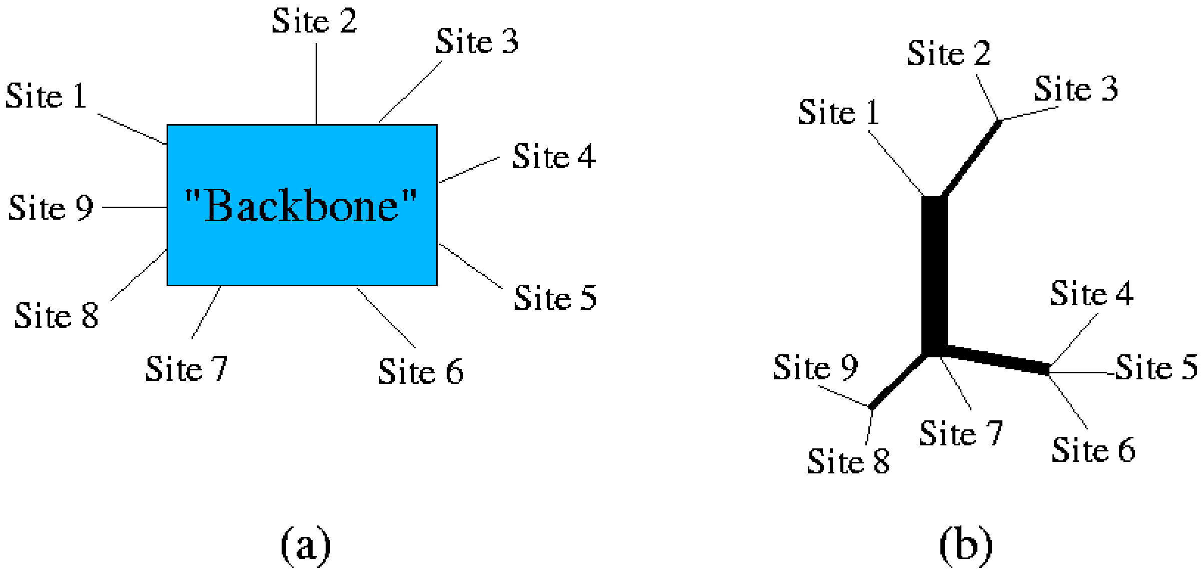

Figure 7: (a) The simple network topology with the Backbone Actor. All geographical sites in the Grid are connected through a single point, and are effectively equidistant.
(b) A more realistic network structure.

through the NIC of the sending SE, it adds the maximum bitrate it will allow for this particular transfer to the message. This information now passes through the Backbone to the NIC at the receiving end. This NIC looks at the maximum bitrate that it can allow for an incoming transfer, and adjusts the figure in the message downward if necessary. It then passes on this message with a finalised bitrate to the SE receiving the file, and sends a confirmation message back to the sending NIC to confirm the details of the transfer.

The details have now all arrived at the receiving SE, and it is this Actor that controls the transfer. Using the bitrate and file size figures it has been given, it calculates when the transfer will finish, and places an event in the DE event queue. When the transfer is complete, it sends a message back down the chain to inform all Actors concerned with it. If there is a change in the bitrate during transfer, a new time is calculated.

In principle this structure should still work if a more sophisticated network model replaced the Backbone. File transfer details could still be passed around in this way, but a more realistic handling of other bottlenecks between the two sites could further modify the transfer rate before it reaches the receiving SE.

This flow of events is shown in fig. 8.

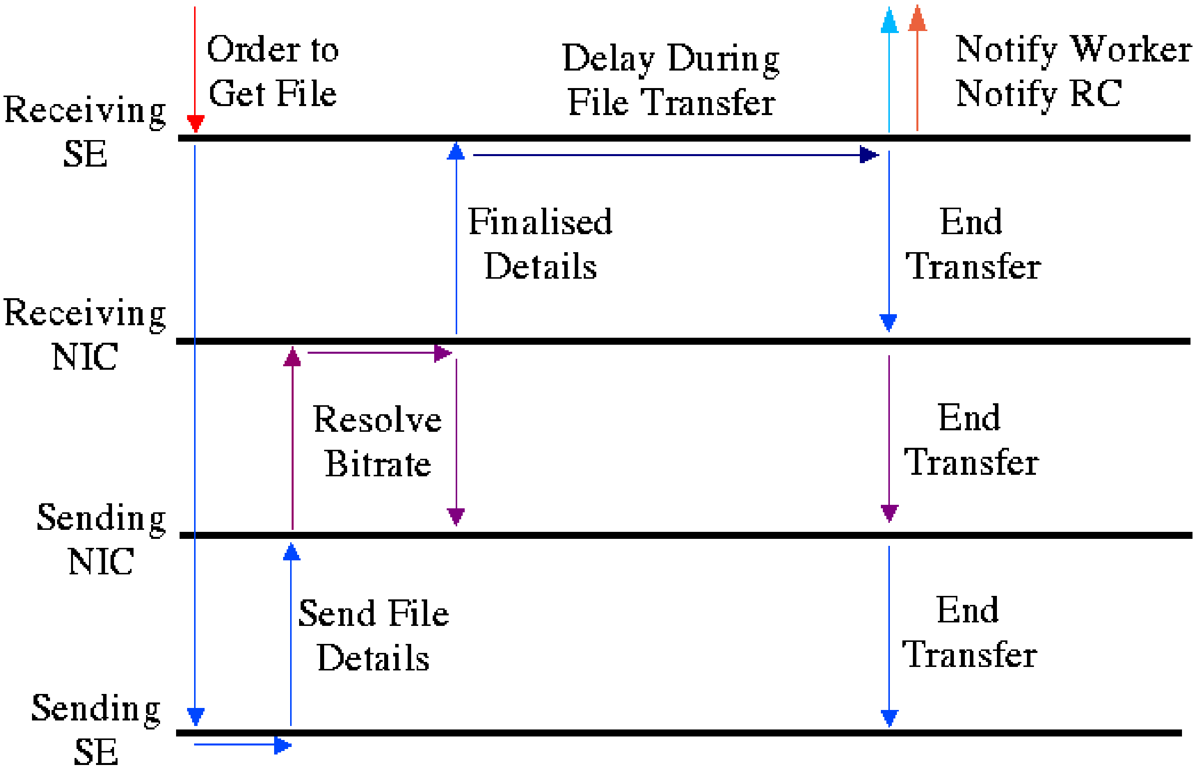

Figure 8: Communication between Storage Elements and NICs to simulate file transfer

6.2.2. User Interface

The User Interface is the Actor that generates the jobs for the simulated Grid to process. These Job objects have a certain memory requirement, need to run for a certain number of CPU cycles, and can have a requirement for input data from one of the SEs. Job generation is triggered by a token arriving at one of the UI's ports. The job is sent through the network to the Resource Broker for scheduling through another port.

The UI can be modified to change the distribution in time or size of the jobs generated, in terms of how much memory it requires and how many CPU cycles it will need to run. The size of the dataset can also be specified, and the dataset will be chosen randomly from the files distributed between the Storage Elements in the simulated Grid.

6.2.3. Resource Broker

When the job arrives at the Resource Broker, the first thing it looks for is the Requirements list in the job's simulated JDL and produces a list of the resources which match them. Currently this will be all available resources, as the Worker machines in the sim are generic, so all jobs can run on any machine.

Next, the RB checks for any data input requirements. If so, it takes any requested logical file names and sends them to the Replica Catalog. The job will wait at the RB until the resolved PFNs arrive back from the RC, when the Rank phase of the matching is carried out.

The job is sent to the chosen resource, along with the list of most accessible physical versions of required input data. The RB acts as the JSS as well in the simulation for simplicity's sake. As the JSS must run at the same site in a real EDG setup, this is a reasonable simplification.

6.2.4. Replica Catalog

This is the simplest Actor as the simulation currently stands. It merely stores any information about permanent and replica files, and can resolve logical file names into physical file names. The LFN of a file is the name of the original permanent file. When a replica is made, it is given a new PFN consisting of the LFN prefixed with the name of the SE it is cached on.

When the RB requests LFN resolution, it returns the appropriate list or lists of PFNs. At the moment this is its only function, as there is no independent Replica Manager, as in the real EDG.

6.2.5. Compute Element

Another relatively simple Actor, the Compute Element is responsible for assigning any jobs it is sent to its member Worker machines. It implements a simple queueing routine, in which no more than a single job may run on a given Machine, and jobs are queued until a Machine is available.

It is also responsible for sending regular information updates to the Resource Broker. It can also send these updates on demand. Thus the CE acts as a Producer, but there is no intermediate Consumer / Producer entity currently - the information is sent directly to the Resource Broker.

6.2.6. Worker

The Worker Actor is built around a Vector object, in which jobs currently running on it are stored. It has a speed in CPU cycles per second, and a certain amount of memory available for use by running jobs. It can run multiple jobs simultaneously, and will begin paging to disk if its memory is exceeded. CPU time is shared equally between jobs if there is more than one running. However CE scheduling policy normally restricts the number of jobs to one per Worker at one time.

Each job has a member variable representing its state of completeness, and another representing how efficiently it is running. The former counts down from the total CPU cycles required by the job to zero as the job runs. The latter currently can have two values - 0.0 or 1.0, so that the job is either running or not. The job will not run if none of its required data is present on the local SE, and will wait until it arrives.

When a job begins running, the time at which it will be completed is calculated based on the CPU speed of the Worker machine, and the number of CPU cycles required by the job. A request for the Worker to be woken up at this time is placed in the event queue.

If, under a less strict CE scheduling policy, another job arrived on the Worker while the first was still running, the Worker would now only be able to devote half the CPU time to the first, invalidating the event placed in the queue earlier. In this case, the arrival of the second job would prompt a new, updated event to be requested. When the previous event activates the Actor, it checks for completed jobs, sees that there are none, and becomes dormant again until the second event occurs.

When a job is completed, its results are sent in the form of the same job object that has been running, but with extra information relating to its running (time started, time finished etc.).

6.2.7. Storage Element

The Storage Element is by some way the most complex part of the simulation. It has a set of permanently hosted files, and a cache in which replicas can be stored when required by jobs on a local CE. Requests for file transfer can come from two different sources.

When a job is initially sent to its chosen CE by the Resource Broker, the details of the files that must be acquired for it is sent with it. At the same time that the job is sent to the CE, requests are sent out to the SE local to the CE to acquire the files. These requests will only be fulfilled if the SE is not too busy, i.e. there is room in the cache, and no queued requests for file transfers.

A second request for these files will be sent by the Worker that has been assigned the job. This request is treated as being more urgent as the job is unable to run until the file arrives. Thus if the cache is full, this request will be queued and file transfer initiated as soon as possible. When transfer has completed, a message is sent to the appropriate local CE, which passes it on to the Worker with the job, which can then start running.

The SE keeps track of file transfers with information about bitrate given to it by the local NIC as described earlier. The completion of transfers is handled in a similar way to that in which the Worker deals with job completion. Transfer is initiated and controlled by the SE receiving the file, and the SE that is sending the file need only be concerned with the fact that the file is open and being read.

When a new replica is placed in the cache, or an old one is removed to make way for another, the Replica Catalog is notified. Obviously a replica will not be removed if the file is open and being read by a job or copied to another location, and the replica will be held in the cache until it has been read by at least one job (i.e. the one that requested its transfer).

6.3. Running the Simulation

When the simulation runs, it requires some input data. A seed must be supplied for the random number generator that produces the sequence of jobs from the User Interface. There is a plain text file for each Storage Element, which consists of a list of names representing the permanent files for that SE. By convention these names consist of an indicator of their host SE and a number, for ease of debugging when determining how the files have moved around the Grid. When the simulation starts, these files are read in by the SEs and their permanent disks are populated.

When constructing a simulated Grid in the Vergil GUI, many components require some information when set up. This can be entered simply in a dialogue box. All CEs and SEs require a name, for instance, so that they can be distinguished, and the sites also have a name, attached to the NIC Actor. In combination, the name of a component and the name of its node act like a domain name for that component, allowing messages to be sent to the correct destination.

Other values added in this way include the size of an SE's temporary cache, the CPU speed and memory of a Worker machine, the location of the Resource Broker that a CE is registered with, and so on.

During the initialisation phase of the simulation, registration messages are passed around, so that CEs find out how many Workers they own, the RB is informed of its registered CEs, and the Replica Catalog is also informed of the initial contents of the SEs. The LFNs of all available files are also passed to the UI, so that it can generate jobs with input data requirements.

Once a job has been generated, assigned, queued, given its input data, run, and completed, it produces results. These are sent back to the Backbone Actor, which writes them out to a plain text file, with each line corresponding to a job. The User Interface keeps generating jobs for a fixed period of time, and then the simulation keeps running until all of the jobs have finished and sent their results back.

6.4. Simplifications and Limitations

This is a simulation of the workflow in an EDG setup, representing jobs and data files as they move around the Grid. It is not intended to simulate all of the EDG components in precise detail, as this would lead to code almost as complex as that which is being simulated. Instead the key functionality of the EDG software is represented, and also the major bottlenecks in the flow of data. Simplifications have been made where appropriate.

6.4.1. Simplified File Transfer

Possibly the biggest of these simplifications is the network model using the Backbone Actor, as described earlier. There is no sense of geography in the simulation as it currently stands, and in a way it is a network backbone that is not being represented here. The bandwidth between two sites is based purely upon the maximum bitrate that the sites at either end of the connection can manage. Thus the transfer is affected by other transfers taking place from the sending node and into the receiving node, but not by other transfers that in reality could be sharing a stretch of cable somewhere halfway along the connection.

A consequence of this is that realistic decisions on which replica of a particular data file should be copied to a given location cannot really be made. There is nothing to choose between two candidate SEs, as the connection to them is topologically identical.

Another simplification that has been made in the operation of the networks is that the bandwidth of a file transfer can only decrease, even if a greater bitrate becomes available at one of the communicating sites during the transfer. This may not seems like an obvious decision, but it was made for simplicity's sake, so that the network model did not become too great a part of the project.

When a new transfer starts, the bitrate available to existing transfers out of the sending site, and into the receiving site, must be reduced. Messages must now be sent out to all parties involved in these other transfers to ensure that they are aware of this reduced transfer rate, particularly the receiving Storage Element Actor which must recalculate the completion time for the transfer. This already creates a large communication overhead.

With a number of transfers being reduced in size in this way, the maximum bitrate available to another group of transfers will now increase. Another (even larger) set of messages would have to be sent out to rearrange these transfers, presumably reducing the bitrate available to yet more transfers, and so on. For this reason the only adjustments made are decreases in immediately affected transfers.

Similarly when a transfer is completed, the bitrate it was using is freed up. The chain of readjustments caused by this is not represented in the simulation either.

This leads to a model that is more pessimistic than the reality would be, but this is a small price to pay for the savings in the speed of running of the simulation.

It is also worth noting that the temporary file cache begins each run of the simulation empty, which potentially leads to "end effects" in the run, at least until the cache fills up with transferred files. Any effects from this empty cache can be minimised by running the simulation for longer periods of time.

6.4.2. Simplified "JDL File"

As described earlier, the user is provided with the opportunity to submit a set of preferences as to the type of resource for a given job to run on. This can be anything from a preference for a site with a large number of processors, to a complex expression with weightings on various desired attributes. This allows the Resource Broker to create rankings of candidate resources, and choose the most favourable.

This system is not implemented in the simulation, as doing so would be an unfavourably large task. Not only would the Resource Broker need to have a reasonable facsimile of the Condor engine for resolving these ranking expressions, but the User Interface would also have to become much more complicated. It would need to be able to generate a ranking request for the jobs it generates to put in the JDL file sent with the job. One possibility would be to supply the UI with a set of pre-written Rank expressions to choose from randomly, but as it stands the jobs are submitted and scheduled with identical ranking policies.

The current setup also only has one UI, which sends jobs directly to the Backbone Actor, to pass on to the RB from there. Multiple UIs could be implemented, as in a real Grid, all choosing different ranking preferences. These would still only be simple choices, however, due to the simplistic scheduling abilities of the RB Actor.

6.4.3. Simplified Jobs and Job Running

The jobs that are handled in this simulation are less complex than their real life counterparts. They require a certain amount of memory on the machine that runs them, and they require a certain number of CPU cycles before they are complete. They can run on any machine in the Grid, as there is no concept of differing machine architecture or operating system. They will never fail to complete, so there is no job error handling. Jobs also have no size associated with them for transfer purposes, i.e. they do not take time to transfer. This is acceptable as this time is negligible compared to the time taken for data files to transfer.

There is only one type of file access by a job - if the file is available then the job can run, if not then it cannot. No output is produced by the jobs, meaning the problem of space on an SE for output datafiles, and the registering of these new files, does not arise.

Another problem with a real EDG setup that does not appear here is that of cache space on the Resource Broker. The output will be returned to the RB if the user has not specified an output SE, and this output will remain in the RB cache until retrieved by its owner. Thus if enough users are late in picking up their output from the RB, the cache will fill up and no more results can be returned. This is not the case in this simulation; results are returned by the jobs, but only in the form of information on the job's running that is output to a file.

Queueing on the Compute Elements is quite simplistic. Each worker machine belonging to the CE runs one job at a time, and when a job completes, the next is taken from the front of the queue and sent to the newly free machine. Real CEs can implement a variety of queueing systems to deal with jobs sent to them by the RB.

There is also no limit on the number of jobs that can be submitted to a CE at a time. Real life CEs will specify the number of jobs that they will allow to run, and the total number allowed including those being queued. Every job submitted by the User Interface will be accepted by the Resource Broker and assigned to a CE, no matter how many jobs are already queued there. Thus the input job load must be tailored to the capabilities of the Grid by trial and error under this regime. A better system would allow the RB to reject a submitted job if the Grid is unable to deal with it at the time of submission.

7. Performance of Scheduling Algorithms

The assignment of jobs to resources in the real EDG Resource Broker is carried out by matching job and resource ClassAds, as described earlier. This simulation does not have a ClassAd matching engine, but its RB can be equipped with a variety of algorithms to make intelligent (or less intelligent) decisions on where the jobs go.

These algorithms are roughly equivalent to the Rank phase of the matching process. There is no real Requirements phase, as there is no concept of machine architecture, operating system, etc. The Grid is also homogenous, for reasons to be discussed later. The two main differences between the simulated matching and the real one are that a single criterion is used to assign the job, and the decision is made by the RB with no input from the user.

The type of scheduling for a job will be placed between two broad tendencies, to schedule for the benefit of the job, and to schedule for the benefit of the system as a whole. The former tendency places the job on the resource that will get it completed as quickly as possible, at the possible expense of less efficient use of the system, whereas the latter will place the job so as to share the load evenly across the system, not necessarily benefiting the job itself. In some cases both of these policies might lead to the same decision, but it is unlikely that both can be optimised simultaneously.

The following sections describe the algorithms used in experiments with the simulation.

7.1. Estimated Traversal Time

This algorithm is intended to select the resource for the job that will complete it in the shortest time, based on the times it has recently taken for the available resources to complete jobs. The traversal time is the time from the job's arrival at a Compute Element, until completion and the return of results. It is given by:

$$\text{Traversal Time} = \text{Time Queued} + \text{Time Waiting for Data} + \text{Time Running} \tag{1}$$

To generate the Estimated Traversal Time from this quantity, it is normalised with the queue length of that CE as follows:

$$\text{Estimated TT} = \text{Last TT} * \frac{\text{Current Queue Length}}{\text{Queue Length At Arrival}} \tag{2}$$

Estimated traversal time cannot be calculated until jobs have been completed, and with large jobs this could be some time into a run of the simulation. Until this quantity can be calculated, a round-robin algorithm is used, as described below.

The normalisation with queue lengths means that if the current queue length is greater than that when the last job arrived, the estimate of traversal time can be scaled accordingly. This implies an assumption that the job's lifetime is dominated by the time it spends in the queue. This algorithm ought to perform well in a scenario in which this is the case.

This approach is not obviously the best solution, as it does not take into account the CPU time required to run a given job, so a CE that has just returned the results of a small job will have an erroneously small value for its estimated traversal time. However, it is the default value for the ClassAd Rank attribute, i.e. this quantity is used to schedule the job if the user has not specified any preferences. Thus it is relevant to include this algorithm in the experiment. It falls into the category of job-centred scheduling, as it is trying to find the resource that will complete the job the fastest.

7.2. Location of Data

When jobs are generated by the User Interface, they can be given a required set of input data consisting of one or more files. These will be chosen at random from the files available on the system. When a job is assigned to a CE, any files not already resident on the local Storage Element must be transferred to it.

An algorithm has been implemented which finds the SE with the largest number of the required files in its permanent storage, as opposed to its temporary cache, and sends the job to the CE at that site. In the case of a tie between two or more sites, a random choice is made between the CEs.

This is intended to favour the system at large, by reducing the cost to the network, as fewer files should be moved around. The resource selection is only based upon permanently hosted files, as including replicas as well produces unrealistic behaviour. This is due to a known limitation in the simulated Storage Element as it currently stands. It might be interesting to compare results with and without the replicas taken into account, as by the time a job is finally assigned to a machine to run, any replicas present in the local cache while the job was being scheduled might well have been removed to make way for others.

7.3. Queue Length

This algorithm looks at the current queue lengths on all of the available CEs, and sends the job to the CE with the shortest queue. A random choice is made if there is a tie.

Again this will be favourable to the Grid as a whole, as it spreads the load between all of the sites, ensuring that the job distribution is fair. In the case of a homogenous Grid, this will also favour the jobs, as they will be assigned to a machine and run as quickly as possible. However in the limit of an extremely heterogenous Grid, this will not necessarily be the case. The shortest queue might be at a CE with smaller and less powerful resources than those of a CE with a longer queue. A job could conceivably lose the time saved when queueing by running on a slow machine. Thus the scheduling decision would be favourable for the resource owners of the Grid, but not for the user.

7.4. Location of Data + Queue Length

Both of the above algorithms will lead to situations in which there is a tie between CEs, and a random choice must be made. In order to reduce the frequency of random choices, this algorithm combines the two. First of all, a short list of sites close to SEs with the greatest number of required files is drawn up. This list is then further reduced by choosing the CE or CEs with the shortest queue. Finally, a random choice will be made if more than one CE is left in the list.

7.5. Random

For the purposes of comparison, a scheduler was devised which makes an entirely random choice as to which CE a job is sent to, uncorrelated with the current status of the CEs or any previous decisions made. This should be the least effective scheduler, as any attempts to regulate the workload of the Grid should give a better performance.

7.6. Round Robin

The round-robin approach is to assign jobs to Compute Elements in turn, and then going back to the beginning of the list when the end is reached. This is the least intelligent 'fair' distribution of jobs, as it does not take into account any of the properties of the incoming job, or the attributes or current status of the resources available.

Any algorithm that makes an intelligent choice of resource for the job could be hoped to perform better. If an algorithm seeks to improve job performance, it should process jobs faster than the round-robin algorithm; similarly, an algorithm concerned with system performance should also do better on its own terms.

8. Metrics to Measure Grid Performance

The Grid's performance is not an easy thing to measure, as it is made up of many entities that may work together, but function independently. As mentioned in the previous section, the performance of the jobs and the system are related in a way that is not easy to define, and it is not necessarily possible to optimise both at the same time.

The Grid's performance can be seen in two ways. The user will tend to see things from the perspective of job performance, interested largely in how quickly or efficiently the results are returned. Conversely, the owners of resources will be looking at how fairly and efficiently those resources are being used. A decision must be made as to which of these is more important, as a trade-off will have to be made.

Below are a selection of metrics that can be used to measure performance, from one or the other of these perspectives.

8.1. Completion Time

In this simulation, a reproducible set of jobs can be generated, and Grid performance can be tested under different conditions with the same distribution of jobs. These jobs are generated for a certain period of time, and the simulation then runs until all jobs have been processed.

This means that the time taken for all of this to happen is given by the time at which the last job to be completed returns its results. Different conditions will give different values for this quantity, as any backlog of unprocessed jobs is dealt with.

8.2. Network Usage (Number of Transfers)

Any time a file is copied from one location to another to make a new replica, there is a cost in terms of network usage. This cost will vary depending on the extent of this network usage. The cost to the network, in terms of the distance the file is transferred d, the rate of transfer r, and the time the transfer takes t, could be measured as follows:

$$\mathrm{C_N} = d * r * t \tag{3}$$

However the concept of distance here is not relevant, due to our simplified network. The product of the transfer rate and the time taken for the transfer is simply the file's size, which as a reasonable approximation can be set to be the same. In this situation, the network cost becomes the same for all transfers.

Thus for this simulation, the total network cost is simply the number of transfers that have been made during a given run.

8.3. Machine Load

The average machine load represents the proportion of a given period of time that a CPU has been active, i.e. running a job. This will be a number between 0 and 1. The period of time represented will be the time elapsed since the last value was given.

This is a measure of how well the machines across the Grid are being used. If they are being effectively utilised, the average machine load will be closer to 1.

8.4. Queue Length

As jobs are submitted to a CE, they will be assigned to machines as they become available. A queue will develop if jobs arrive faster than they can be processed. Such an excess of jobs may develop if a resource scheduling algorithm is not distributing the workload effectively.

This means that for a reasonably loaded Grid, the queue lengths of its CEs, i.e. the number of jobs in the queues, are a measure of how effectively the computing resources are being used.

8.5. Traversal Time

This is simply the traversal time of the job, from its arrival at the CE to its completion, as described earlier. This is a measurement of performance on a job-by-job basis.

9. Resource Allocation in a Homogenous Grid

Even in a simplified Grid, such as that represented in this simulation, there are a great many variables affecting its behaviour and ability to process jobs. In order to reduce the number of these variables, the first set of experiments was carried out using a homogenous Grid. This configuration is unlikely to occur in reality, but it can demonstrate certain aspects of the behaviour of an EDG setup.

Ten nodes were created, consisting of a composite object containing all of the Actor objects for that node. Each composite Actor represents a physical site that is a member of the Grid, and the Actors contained by it represent the entities at that site, e.g. Compute Element, worker machines, etc.

At each site was one Compute Element, managing ten Worker machines each. These Workers were also identical in terms of memory and CPU speed (nominally 64MB and 166MHz, although the numbers have little importance in this homogenous Grid). Each site could manage a maximum of 0.01 GB/sec in total for file transfers, both inward and outward. Each of these sites was also given one Storage Element, with fifty 1GB files permanently stored on each, and 200GB of cache space for temporary files.

The Resource Broker and Replica Catalog were located at the same site. With the simplified network model, their location is not significant, as all communication other than data file transfer is effectively instantaneous. In a more realistic Grid, these entities might be located at different sites, and there might be communication issues to consider.

9.1. Moderate Load, Equal Job Sizes

Given a homogenous Grid topology and a moderate load of equally sized jobs, the performance of six different scheduling algorithms was tested. The algorithms used were those described in chapter 7, and the metrics used to measure their performance were those described in chapter 8.

For the first experiment, the sizes of the jobs generated by the User Interface Actor were set to be equal. This is not an unreasonable approximation, and was in keeping with the idea of a homogenous Grid. They each required 50 000 seconds to run over their 10GB of required input data, and 32MB of system memory to run (although with only one job per machine and all machines having 64MB of memory this has no effect on the running of the simulation). The only difference between jobs was the input data required. This was in the form of ten 1GB data files, randomly chosen from those available in the SEs of the Grid.

They were generated at Poisson distributed time intervals, with an average of 300 seconds. The final job was generated at 100 000 seconds, and the simulation then ran until the last job had completed. This random distribution of jobs was reproducible, in order that the performance of different scheduling algorithms could be tested with the same job load.

With these input parameters, the simulation was run 100 times for each of the six scheduling algorithms. The results are shown in fig. 9 and fig. 10. These plots are summarised in tables 1, 2, 3, 4 and 5.

The performance of the system in terms of job processing can be seen in table 2. Perhaps surprisingly, the round-robin algorithm performs most effectively, with the estimated traversal time algorithm close behind it. One might be surprised, as this algorithm is static, not making any decisions based on dynamic information returned by the resources, such as the queue length algorithm makes.

Similarly, table 3 shows that the round-robin algorithm is the most effective from the user's point of view, on a job-by-job basis (although estimated traversal time gives a slightly more consistent performance). The dynamic algorithms are performing less well than the static one.

The two algorithms making decisions based on the location of required input data are not competing well on these terms, but in table 1 we can see that they are reducing the load on the network by around 15 per cent. This would reduce the overheads associated with file transfer. When CE queue length is also considered, the throughput of jobs is improved considerably, although performance still lags behind round-robin scheduling.

Table 4 shows that all of the algorithms are loading the machines across the Grid more or less equally, except for the random algorithm which is performing poorly in all areas, as would be expected. The mean queue lengths given in table 5, related to how quickly the system is processing jobs, shows interestingly that the algorithm based on minimising queue lengths is not producing the best results in this area - again, round-robin and estimated traversal time perform slightly more effectively.

The clue as to why this is happening is in the poor performance of the random algorithm. When the algorithms making decisions based on dynamic information need to resolve a tie between two or more resources, they make a random decision. The round-robin algorithm never needs to do this - it just keeps cycling round the available resources, one at a time.

The other important factor is the homogeneity of the Grid being used, and the identically sized jobs. Let us compare the behaviour of the round-robin and smallest-queue algorithms. The latter has very little to choose between, as it can only see the length of the queues at the various resources. As the arrival rate of the jobs is much smaller than the time needed for the jobs to run (~300 sec vs. ~50000 sec), it is very unlikely that the queue lengths will differ by more than one. In the case that the queue lengths are all equal, the algorithm will pick one at random, with no other information to go on. It cannot tell which site is closest to removing a job from the queue to place on a machine that has completed its previous job.

The round-robin algorithm has no knowledge of these things either, but it sends the job to the resource that it least recently sent a previous job to. It is likely that this is the resource that is closest to removing a job from its queue. Thus the round-robin algorithm has made a more effective decision than the shortest-queue algorithm.

It should be pointed out that this is an unusual circumstance based upon the homogeneity of this Grid setup, and of the jobs being generated. Nevertheless, in this scenario the round-robin algorithm is making more effective decisions than the others, despite not making use of feedback from the resources.

The two algorithms that could be said to be competing with it are the ones based upon estimated traversal time, and both data location and queue length information. The former is partly doing better than the others because until the first jobs start returning their results (50000 seconds into the simulation onwards), it is using the round-robin algorithm to make decisions.

The other does not perform as well in terms of last completion time, or job traversal times, but it is not too far off. Its strength is in its reduction of network cost in combination with reasonable job processing. The algorithm that schedules purely based on data location does not perform nearly as well.

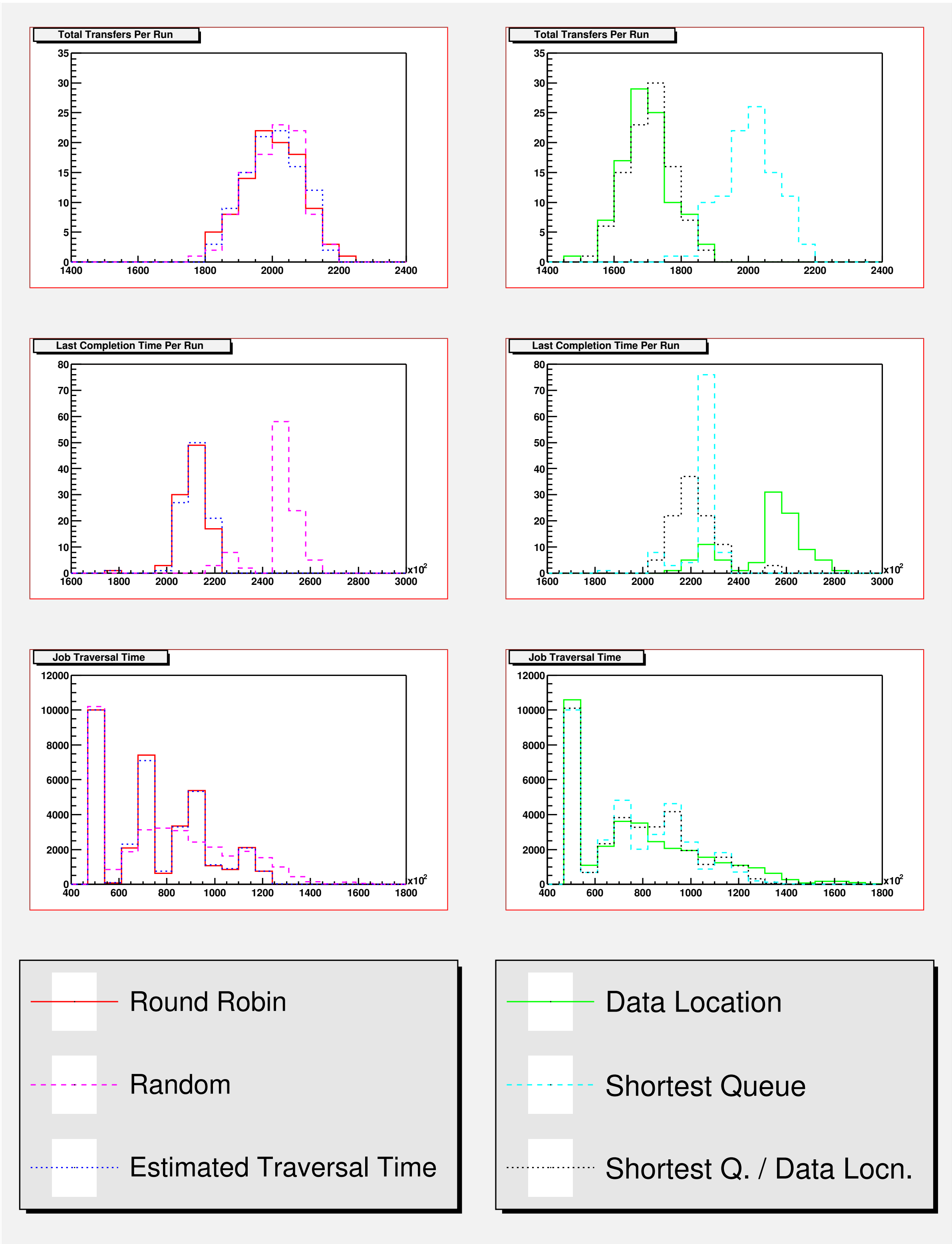

Figure 9: Total transfers per run (top), completion time of last-completed job (middle), and job traversal times (bottom), for six different scheduling algorithms, with a homogenous Grid and a moderate load of equally sized jobs. The algorithms have been split up for ease of comprehension, with the round-robin, random and estimated traversal time plots on the left, and data location, shortest queue and data location / shortest queue on the right.

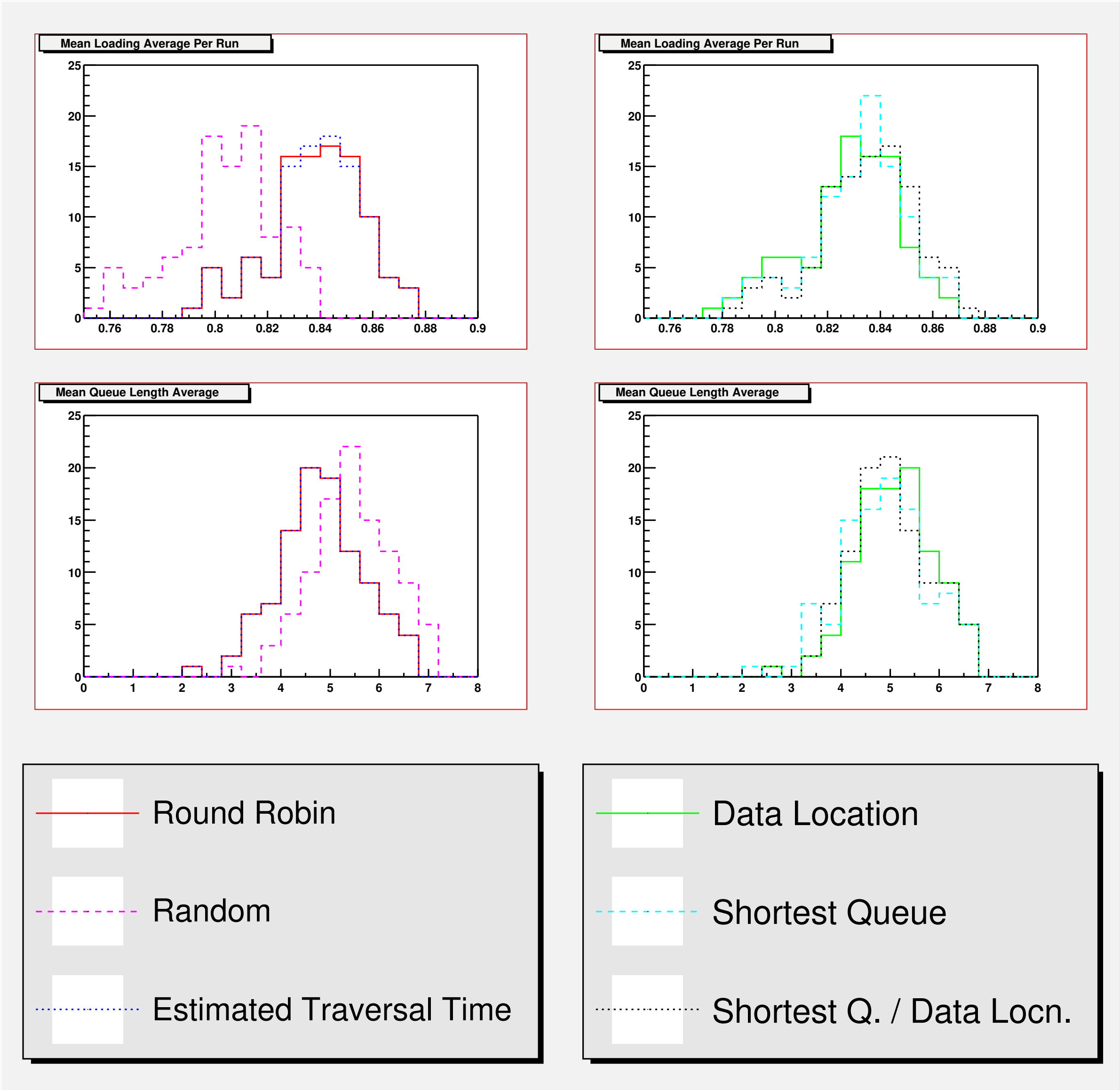

Figure 10: Mean machine loading average (top) and mean queue length (bottom), for six different scheduling algorithms, with a homogenous Grid and a moderate load of equally sized jobs.

Scheduler	Mean No. Transfers per Run	RMS
Round-Robin	2005	85
Random	2004	79
Estimated Traversal Time	2004	82
Data Location	1700	72
Min. Queue Size	2005	85
Data Lcn. / Queue Size	1707	71

Table 1: Number of file transfers in a run of the simulation, with a homogenous Grid and a moderate load of equally sized jobs.

Scheduler	Mean Last Completion Time / 10^5 s	RMS
Round-Robin	2.10	0.58
Random	2.26	0.32
Estimated Traversal Time	2.11	0.57
Data Location	2.26	0.58
Min. Queue Size	2.24	0.73
Data Lcn. / Queue Size	2.20	0.68

Table 2: Last completion time of the last-completed job in a run of the simulation, with a homogenous Grid and a moderate load of equally sized jobs.

Scheduler	Mean Traversal Time / 10^5 s	RMS
Round-Robin	1.03	0.08
Random	1.28	0.10
Estimated Traversal Time	1.05	0.03
Data Location	1.25	0.11
Min. Queue Size	1.09	0.09
Data Lcn. / Queue Size	1.13	0.07

Table 3: Traversal times of jobs, with a homogenous Grid and a moderate load of equally sized jobs.

Scheduler	Mean Loading Avg. per Run	RMS
Round-Robin	0.84	0.02
Random	0.81	0.02
Estimated Traversal Time	0.84	0.02
Data Location	0.84	0.02
Min. Queue Size	0.84	0.02
Data Lcn. / Queue Size	0.84	0.02

Table 4: Mean value of the averaged machine load average over a run of the simulation, with a homogenous Grid and a moderate load of equally sized jobs.

Scheduler	Mean Queue Length	RMS
Round-Robin	4.8	0.9
Random	5.4	0.8
Estimated Traversal Time	4.8	0.9
Data Location	5.1	0.8
Min. Queue Size	4.9	0.9
Data Lcn. / Queue Size	5.0	0.8

Table 5: Mean length of queue over a run of the simulation, with a homogenous Grid and a moderate load of equally sized jobs.

9.2. Heavy Load, Equal Job Sizes

The six previously described algorithms were again tested with a homogenous Grid topology and equally sized jobs, but with a heavier loading of those jobs.

This is essentially the same experimental setup as the previous experiment, but with the average interval between job generations reduced from 300 to 100 seconds, and the time of generation of the last job reduced from 100000 to 40000 seconds. This experiment was carried out with particular interest in the performance of the estimated traversal time algorithm, the default quantity used for ranking in the Resource Broker's ClassAd matching procedure. As it is scaled with the lengths of queues at the Compute Elements, it should perform more effectively in the limit of a job's lifetime being dominated by the time it spends queueing. The higher arrival rate of jobs means that such a situation will occur, as the jobs are sent out more rapidly than the CEs can cope with. The results of this run are shown in fig. 12 and fig. 13.

The most striking thing about the plots of job traversal times and last completion time is the structure that has now appeared, with a series of narrow peaks rather than one broader one as might be expected for the former, and a peak tailing off gradually for the latter. This turns out to be a result of the artificially homogenous Grid, and the fact that the job arrival times are so close together.

Plotting last completion time against the number of jobs in a run of the simulation shows a large discontinuity. This appears to occur around the point corresponding to 400 jobs. This is a multiple of the total number of machines across the Grid, 100, so further runs were carried out with average job intervals of 125, 200, 300 and 400 seconds to see if similar discontinuities occurred at each multiple of 100. As can be seen in fig. 11, this is the case. The scheduler used was the round-robin algorithm, as the discontinuity was most pronounced in its results.

The first values of the last completion time appear sharply at 90000 seconds. This is because the minimum time a run can take is the time at which the last job is generated (40000 seconds), plus the time taken for the job to run, which will be at least 50000 seconds, plus any time needed for file transfer. Below a total of 100 jobs in a run, there are enough machines to run all jobs without a need for queueing, so the last completion times cluster around this point.

The first discontinuity produces a jump to 100000 seconds, and after that point there is a steady increase in completion time with total jobs, broken only by further discontinuities. These are 50000 second jumps, the CPU time required by the jobs. What is happening is that beyond 100 jobs in a run, further jobs must be queued. This means that at least one machine must process two jobs before it is done. At the next discontinuity, some machines must start processing three jobs, and so on.

Of course such an unusual looking set of results could only appear with the extremely homogenous Grid and identical jobs present in this experiment. If one looks at the plots for the previous experiment with a slower arrival rate of jobs, the discontinuities are

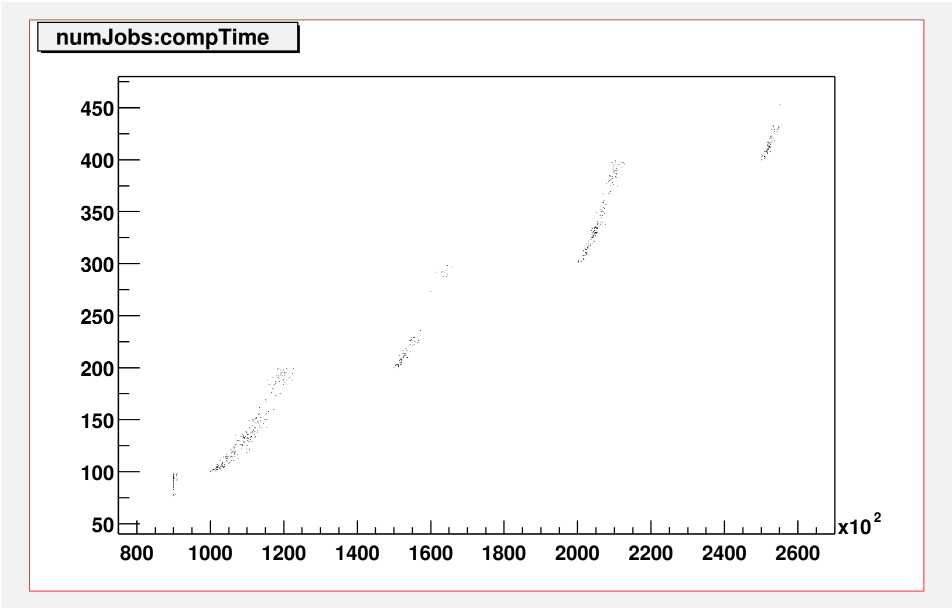

Figure 11: Last completion time of last-completed job vs. number of jobs per run, showing the discontinuities every 100 jobs.

present there too, although they are less pronounced as the jobs are not arriving as rapidly. This phenomenon affects all algorithms, so comparison of their performance is not impaired.

If we look at tables 7 and 8, the estimated traversal time algorithm is now performing as well as the round robin algorithm, from the point of view of the resource owners or those submitting the jobs. Interestingly the algorithm based on data location and queue length is processing individual jobs as well as both of these on average, although it loses out to the queue length only algorithm in terms of overall throughput.

Looking at the other metrics, things are much the same as they are in the previous experiment, with the two algorithms that consider data location reducing the network load significantly. Random scheduling continues to prove less effective than the other methods, as would be hoped.

Although both the estimated traversal time and data location / queue length algorithms have improved their relative performance, the round-robin algorithm is still at least equalling them. In this homogenous Grid scenario, it seems to be more or less the ideal way to share the job load across the resources. A less symmetrical regime needed to be introduced, in which the dynamic scheduling policies could be given the chance to prove themselves.

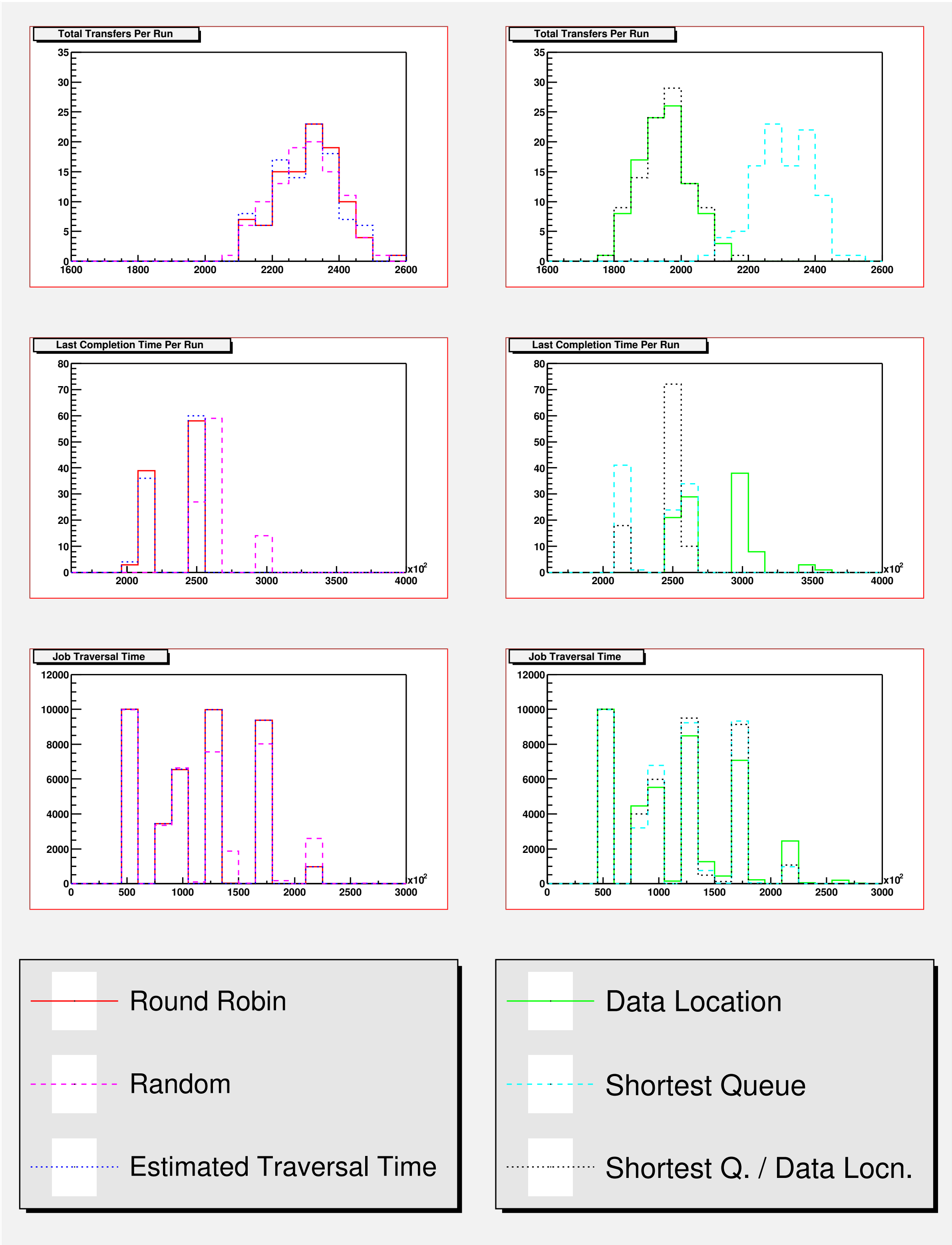

Figure 12: Total transfers per run (top), completion time of last-completed job (middle), and job traversal times (bottom), for six different scheduling algorithms, with a homogenous Grid and a heavy load of equally sized jobs.

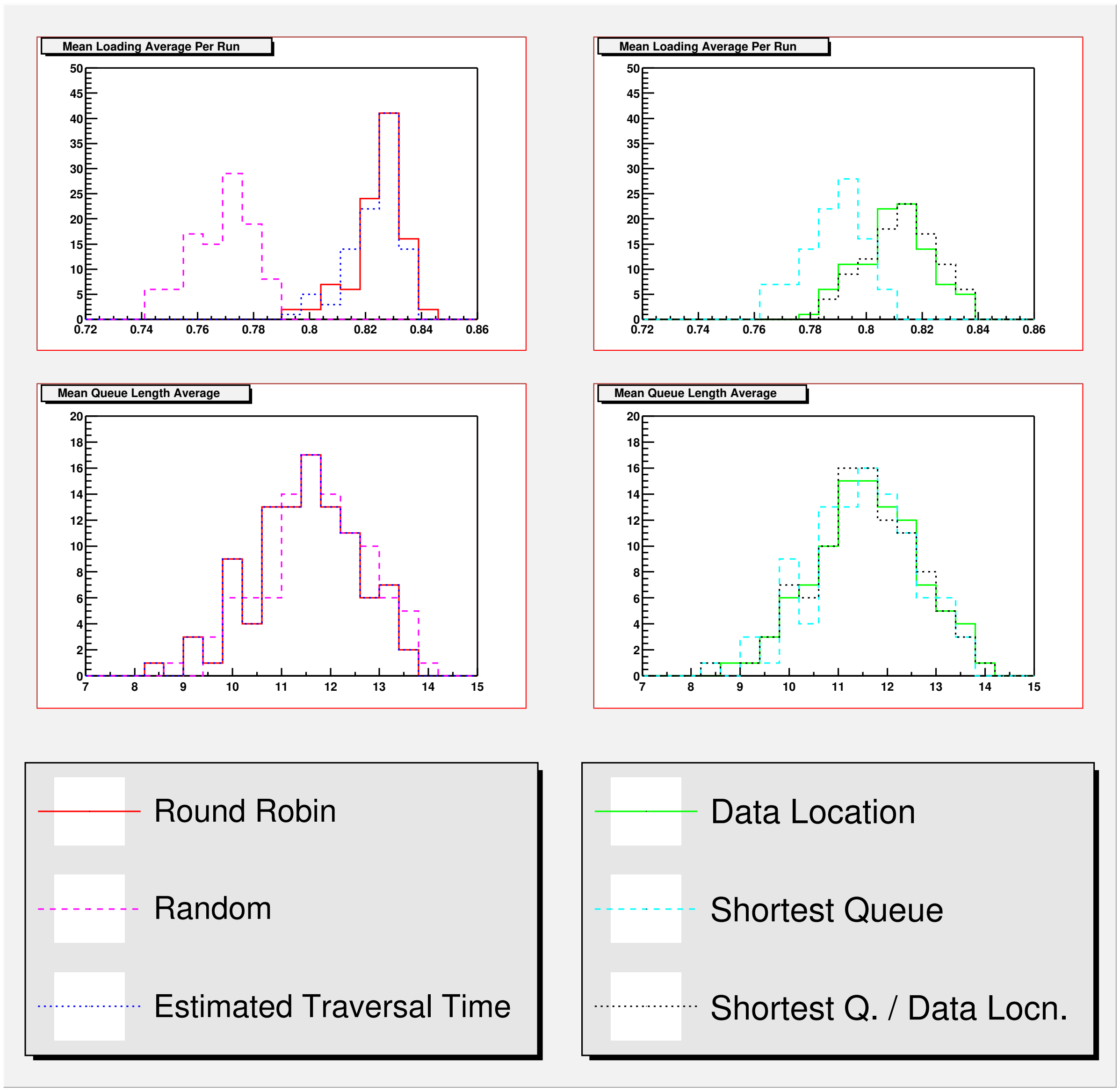

Figure 13: Mean machine loading average (top) and mean queue length (bottom), for six different scheduling algorithms, with a homogenous Grid and a heavy load of equally sized jobs.

Scheduler	Mean No. Transfers per Run	RMS
Round-Robin	2306	93
Random	2300	94
Estimated Traversal Time	2302	95
Data Location	1950	74
Min. Queue Size	2306	85
Data Lcn. / Queue Size	1951	70

Table 6: Number of file transfers in a run of the simulation, with a homogenous Grid and a heavy load of equally sized jobs.

Scheduler	Mean Last Completion Time / 10^5 s	RMS
Round-Robin	2.34	0.21
Random	2.63	0.16
Estimated Traversal Time	2.35	0.21
Data Location	2.82	0.27
Min. Queue Size	2.39	0.20
Data Lcn. / Queue Size	2.46	0.16

Table 7: Last completion time of the last-completed job in a run of the simulation, with a homogenous Grid and a heavy load of equally sized jobs.

Scheduler	Mean Traversal Time / 10^5 s	RMS
Round-Robin	1.12	0.47
Random	1.15	0.51
Estimated Traversal Time	1.12	0.47
Data Location	1.14	0.52
Min. Queue Size	1.13	0.47
Data Lcn. / Queue Size	1.12	0.47

Table 8: Traversal times of jobs, with a homogenous Grid and a heavy load of equally sized jobs.

Scheduler	Mean Loading Avg. per Run	RMS
Round-Robin	0.82	0.009
Random	0.77	0.011
Estimated Traversal Time	0.82	0.009
Data Location	0.81	0.013
Min. Queue Size	0.79	0.011
Data Lcn. / Queue Size	0.81	0.012

Table 9: Mean value of the averaged machine load average over a run of the simulation, with a homogenous Grid and a heavy load of equally sized jobs.

Scheduler	Mean Queue Length	RMS
Round-Robin	11.5	1.1
Random	11.7	1.1
Estimated Traversal Time	11.6	1.1
Data Location	11.6	1.1
Min. Queue Size	11.5	1.1
Data Lcn. / Queue Size	11.5	1.1

Table 10: Mean length of queue over a run of the simulation, with a homogenous Grid and a heavy load of equally sized jobs.

9.3. Moderate Load, Variable Job Sizes

The six scheduling algorithms were tested on a homogenous Grid with a moderate loading of jobs with a variable CPU time requirement.

This variation was introduced as a Gaussian distribution centred around 50000 seconds, the previous value of CPU requirement, and a standard deviation also of 50000 seconds. The distribution was truncated to not fall outside the range 500 seconds to 100000 seconds. The results of these runs are shown in fig. 14 and fig. 15.

It can be seen that the peaked structure in the traversal and last completion time plots has disappeared. The jobs being generated are now taking differing times to be completed, and the discontinuities as the worker machines fill up are smoothed over.

We can immediately see a collapse in the performance of the estimated traversal time algorithm. Table 12 shows that it is doing significantly less well even than the random algorithm, which ought to be less effective than any algorithm making sensible decisions. Similarly, table 13 shows the estimated traversal time scheduler processing jobs much less effectively on an individual basis.

The problem arises because this algorithm is based upon the traversal times of previous jobs, which in this run are unrelated to the size of the job currently being assigned. A resource that has just run a very short (in CPU time) job will return an anomalously small value for the traversal time. If it is some time until the next job at that site completes, every job received by the Resource Broker may be sent to this site, giving it a load beyond its capabilities. The normalisation by queue length data may help a little, but jobs' traversal times are not sufficiently dominated by time spent in the queue for this to minimise this effect.

This means that the estimated traversal time algorithm is making decisions which are actively counter-productive to the functioning of the Grid. Its results are worse than those of the random algorithm, because its choices seriously unbalance the system, whereas the random scheduler can at least achieve some level of fairness in its distribution of jobs.

The round-robin algorithm is still performing well, despite the variation that has been introduced in job sizes. This is most likely to be due to the topology of the Grid itself still being completely homogenous. Any irregularity in load due to the sizes of the jobs assigned to a given site is likely to average out across the simulation runs, or even across a single run, as a site processes tens of jobs.

However, if we look at the traversal time and completion time figures for the minimum queue length algorithms, they have caught up with the round-robin scheduler. The algorithm concerned with queue length only performs very slightly better than the other, as sending the job to the data produces a less even job distribution than the random choice made by the queue length algorithm as a tie breaker.

Again the data location algorithm struggles in comparison, but table 11 shows that it still produces a significant reduction in network use. The data location / queue length algorithm, however, achieves this lower network use, but is still able to compete with the other algorithms which are successful in terms of job processing.

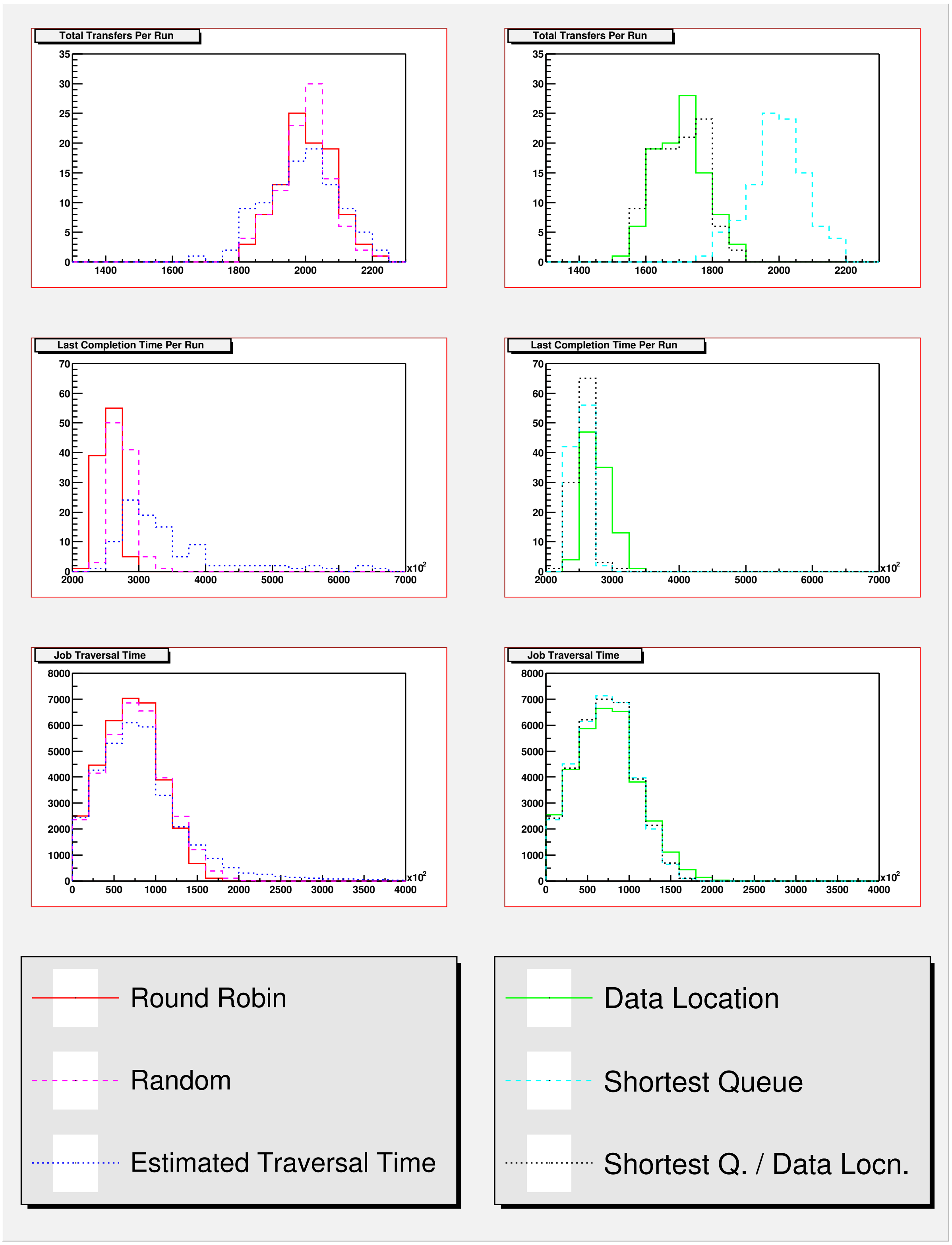

Figure 14: Total transfers per run (top), completion time of last-completed job (middle), and job traversal times (bottom), for six different scheduling algorithms, with a homogenous Grid and a moderate load of variable sized jobs.

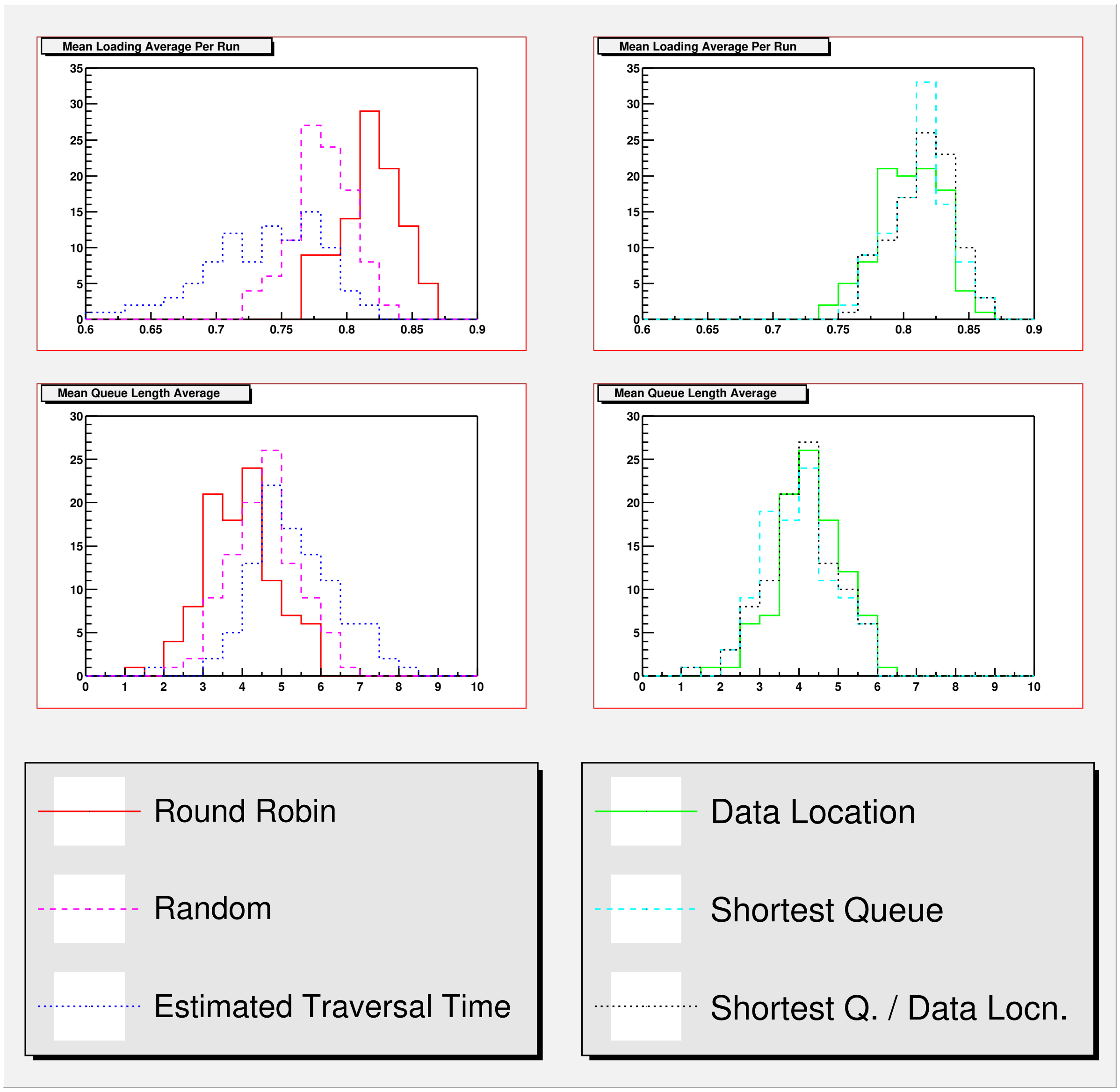

Figure 15: Mean machine loading average (top) and mean queue length (bottom), for six different scheduling algorithms, with a homogenous Grid and a moderate load of variable sized jobs.

Scheduler	**Mean No. Transfers per Run**	**RMS**
Round-Robin	2006	82
Random	2000	76
Estimated Traversal Time	1989	107
Data Location	1705	74
Min. Queue Size	1997	86
Data Lcn. / Queue Size	1706	73

Table 11: Number of file transfers in a run of the simulation, with a homogenous Grid and a moderate load of variable sized jobs.

Scheduler	**Mean Last Completion Time / 10^5 s**	**RMS**
Round-Robin	2.55	0.13
Random	2.74	0.16
Estimated Traversal Time	3.50	0.89
Data Location	2.78	0.18
Min. Queue Size	2.53	0.12
Data Lcn. / Queue Size	2.54	0.12

Table 12: Last completion time of the last-completed job in a run of the simulation, with a homogenous Grid and a moderate load of variable sized jobs.

Scheduler	**Mean Traversal Time / 10^5 s**	**RMS**
Round-Robin	0.710	0.344
Random	0.751	0.376
Estimated Traversal Time	0.843	0.562
Data Location	0.741	0.380
Min. Queue Size	0.710	0.340
Data Lcn. / Queue Size	0.715	0.345

Table 13: Traversal times of jobs, with a homogenous Grid and a moderate load of variable sized jobs.

Scheduler	**Mean Loading Avg. per Run**	**RMS**
Round-Robin	0.82	0.022
Random	0.78	0.022
Estimated Traversal Time	0.74	0.044
Data Location	0.80	0.025
Min. Queue Size	0.81	0.022
Data Lcn. / Queue Size	0.81	0.022

Table 14: Mean value of the averaged machine load average over a run of the simulation, with a homogenous Grid and a moderate load of variable sized jobs.

Scheduler	**Mean Queue Length**	**RMS**
Round-Robin	3.9	0.9
Random	4.6	0.9
Estimated Traversal Time	5.3	1.1
Data Location	4.3	0.8
Min. Queue Size	4.0	0.9
Data Lcn. / Queue Size	4.1	0.9

Table 15: Mean length of queue over a run of the simulation, with a homogenous Grid and a moderate load of variable sized jobs.

10. Stability Over Longer Runs

Using the round-robin scheduler and a moderate load of equally sized jobs with a homogenous Grid structure, the effect of varying the period of time over which jobs are generated was measured.

For this experiment, the same setup as the previous one was used, with jobs generated at Poisson distributed intervals with an average gap of 300 seconds as before. However the period over which these jobs were generated was varied. 100 runs were carried out with job generation periods of 50 000, 100 000, 200 000, 300 000 and 400 000 seconds.

A real Grid would have a saturation point, in which the loading of the resources with jobs would reach a plateau. Each CE would have a limit to the number of jobs they would accept, both queued and actually running on a worker machine. When this limit is reached, no more jobs will be sent by the Resource Broker. At higher and higher loadings, with jobs arriving faster than they can be processed, a point will eventually be reached where submitted jobs are rejected, as the resources to process them are no longer available.

A limitation of the simulated Compute Elements here is that they have no such limits, meaning that they will keep accepting new jobs, no matter how large the load becomes. This experiment was carried out as a check that the simulation behaves convincingly under these loads. The results are shown in fig. 16 and fig. 17. Ratios of the various metrics to the time over which jobs were generated were also plotted in fig. 18.

The results show that the simulated Grid behaves as one might expect. Last completion and traversal times increase in proportion with the load imposed on the Grid. The size of the queues behaves similarly, as it is a direct representation of this loading, and the number of files transferred over for these jobs also increases linearly. The machine load average does not increase in this proportional manner, but it asymptotically approaches a completely loaded state. This is because the period at the beginning of the run while the machines fill up with jobs gets relatively smaller and smaller as the runs get longer. The mean was taken over the period during which jobs are being generated, so the gradual tail-off as the jobs are processed is not included in this calculation.

Overall, this experiment shows that the virtual Grid is behaving realistically when given a high workload to deal with. However an improved model would give the CEs the ability to limit the job load that they would accept.

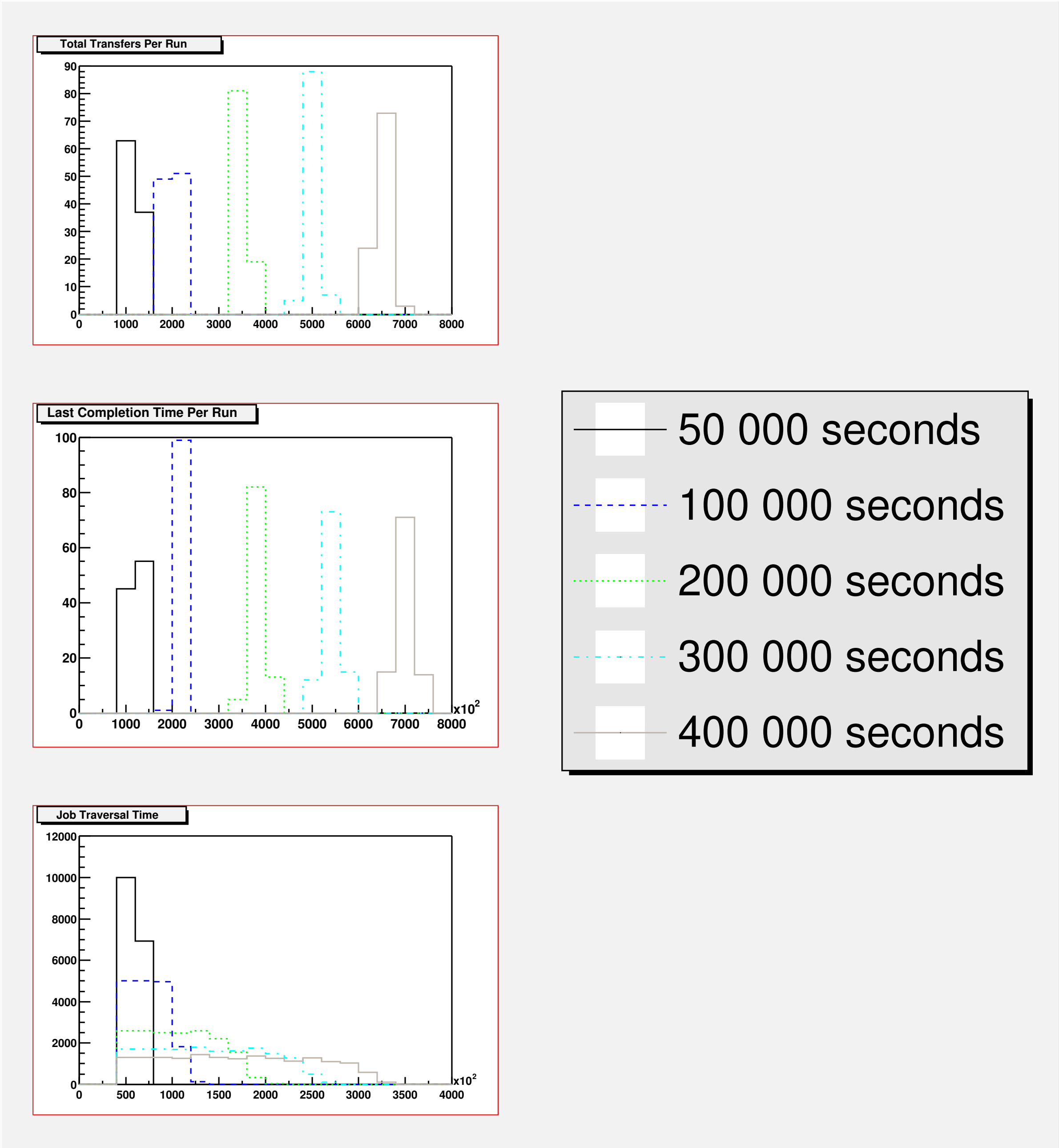

Figure 16: Total transfers per run (top), completion time of last-completed job (middle), and job traversal times (bottom), for five different lengths of job generation time.

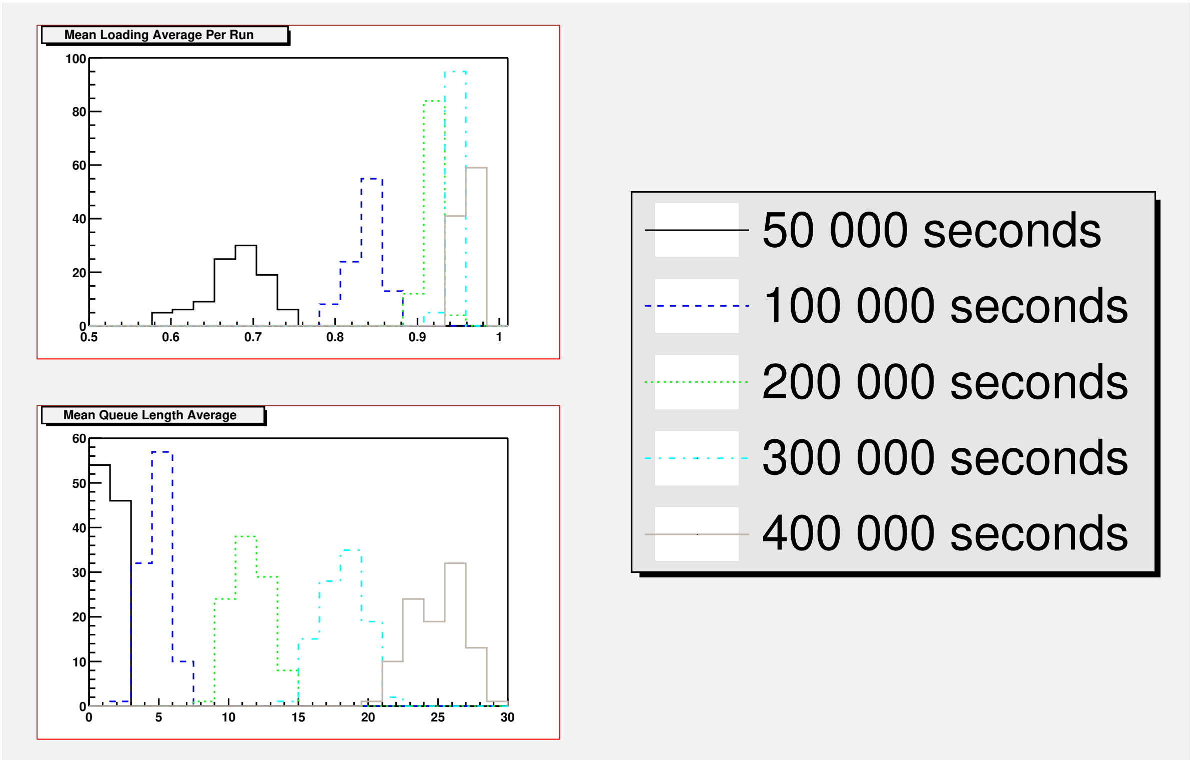

Figure 17: Mean machine loading average (top) and mean queue length (bottom), for five different lengths of job generation time.

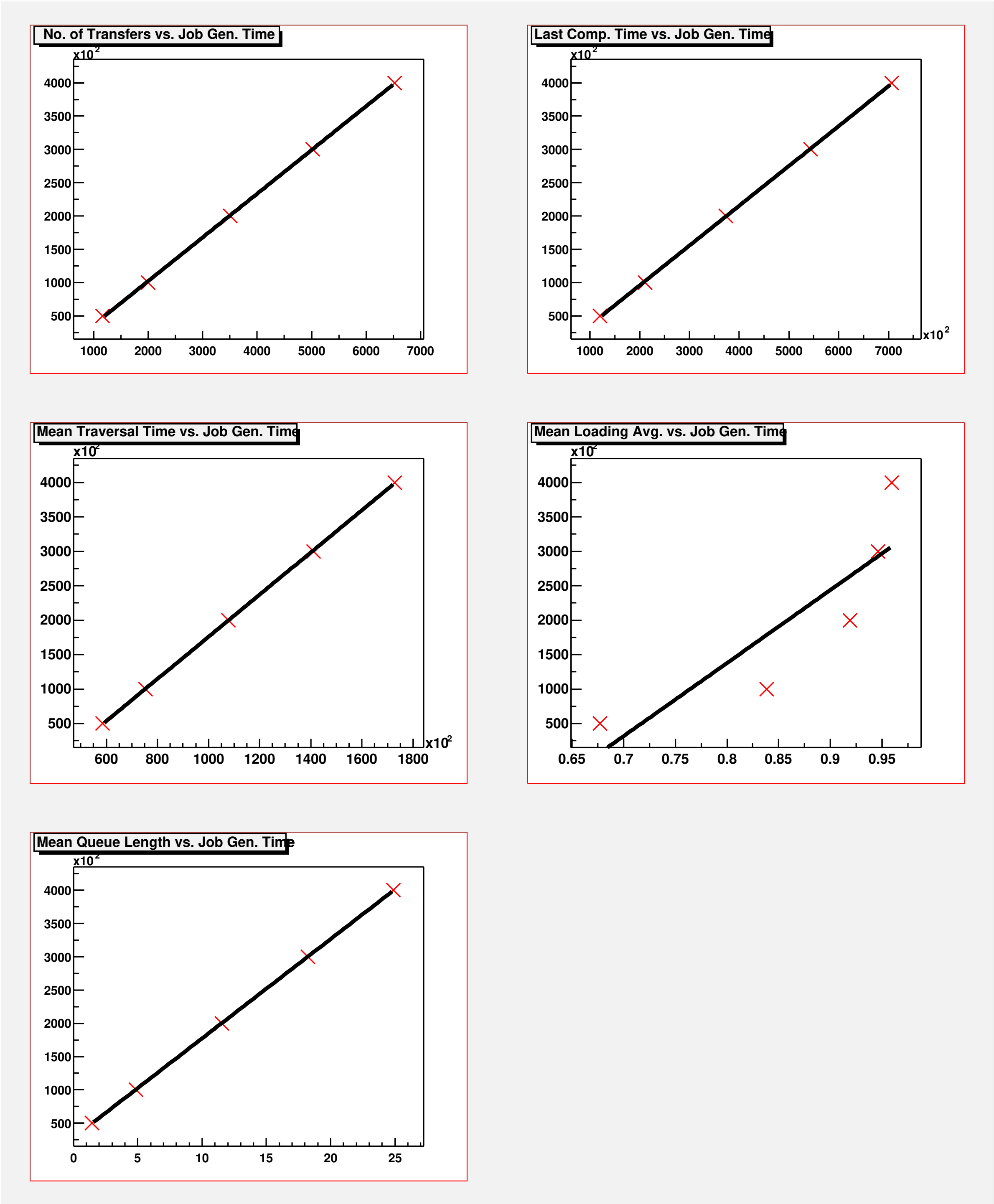

Figure 18: Plots of the mean of five performance metrics against job generation time: Number of transfers, last completion time of last-completed job, mean traversal time, machine loading average, and queue length.

11. Resource Allocation in a Heterogenous Grid

Up to this point, the static round-robin scheduling policy had been unrealistically dominant, providing an effective method of load sharing in a homogenous Grid. An alternate Grid setup was devised with similar total resources available to the user, but with a heterogenous topology. Much like the Tier structure of Regional Centres, it was decided that this should have one large site with many machines and sizable data storage facilities, a few medium sized sites, and then several smaller ones.

The total resources in the homogenous Grid were conserved: this consisted of (at each of the ten sites) ten worker machines, 200GB of disk cache, and 50 1GB data files. A new Grid consisting of ten sites was constructed, using this ratio of 20GB cache and 5 data files per machine. The other resource consideration was that of the connection to the network, i.e. the total bitrate of information that the site could pass in or out. The big site with the largest number of data files was given 0.1 GBytes/sec, whereas the others were left with 0.01 GBytes/sec.

Sites	**No. Machines**	**Cache Size**	**No. Data Files**	**Max. Bitrate**
Site 0	40	800	200	0.1 GBytes/sec
Sites 1-4	10	200	50	0.01 GBytes/sec
Sites 5-9	5	100	25	0.01 GBytes/sec

Table 16: Resources of different "Tiers"

11.1. Equal Job Sizes

The six scheduling algorithms were tested on the heterogenous Grid described above, with a moderate load of equally sized jobs.

This is essentially the same experiment as in section 9.1, but with a different Grid topology. The results are plotted in fig. 19 and fig. 20.

The plots of last completion time and traversal time show that the discontinuities present in the previous experiment have reappeared. Although the Grid topology is no longer symmetrical, its worker machines are all still identical, and the CPU requirements of the jobs are also the same. This was the cause of the discontinuities previously, hence their presence here.

The summaries of these quantities in tables 18 and 19 show how the scheduling algorithms have coped with the new topology. The round-robin scheduler is no longer performing well, as its policy of passing jobs one at a time to the member sites leaves Site 0 with its 40 workers underloaded, while the smaller sites with only 5 machines struggle to deal with the workload which they are assigned. The random algorithm is once again producing the least impressive (and most erratic) results.

The two algorithms that send jobs to the largest concentration of data are falling far short of anything other than the random scheduler. This is because the distribution of data is in proportion with that of the resources, and 40% of the data is located at Site 0. Table 20 shows that the data location algorithm leads to an average machine load of 0.40, suggesting that only 40% of machines are loaded at a time, corresponding to the 40% of machines loaded at Site 0.

The algorithm that also takes into account the length of queues fares slightly better, with a load average of 0.47. Occasionally another site will be hosting as many required data files as Site 0, and the job will be sent there, as its machines are likely to be almost idle. The job processing time data confirms that this algorithm is producing better results than the scheduler only concerned with data location, but the improvement is only a small one.

Table 17 shows that this improvement in job processing times is offset by heavier network use. This is because the algorithm selecting data location only almost always sends the jobs to Site 0 which has a bigger cache, and thus is likely to have even more of the required data than the algorithm is picking up on. The more sophisticated algorithm manages to share the load a little better, but the jobs are sent to sites with less files cached, leading to more file transfers. However both algorithms give a much smaller network load than the others. This effect is even greater than in previous experiments, again due to the concentration of data at the most used site.

The best results are produced by the estimated traversal time and minimum queue length algorithms, with the former having a slight edge in performance. They also require less file transfer than the round-robin and random algorithms, because they are sending more jobs to Site 0, in proportion with the available resources.

Figure 19: Total transfers per run (top), completion time of last-completed job (middle), and job traversal times (bottom), for six different scheduling algorithms, with a heterogenous Grid and a moderate load of equally sized jobs.

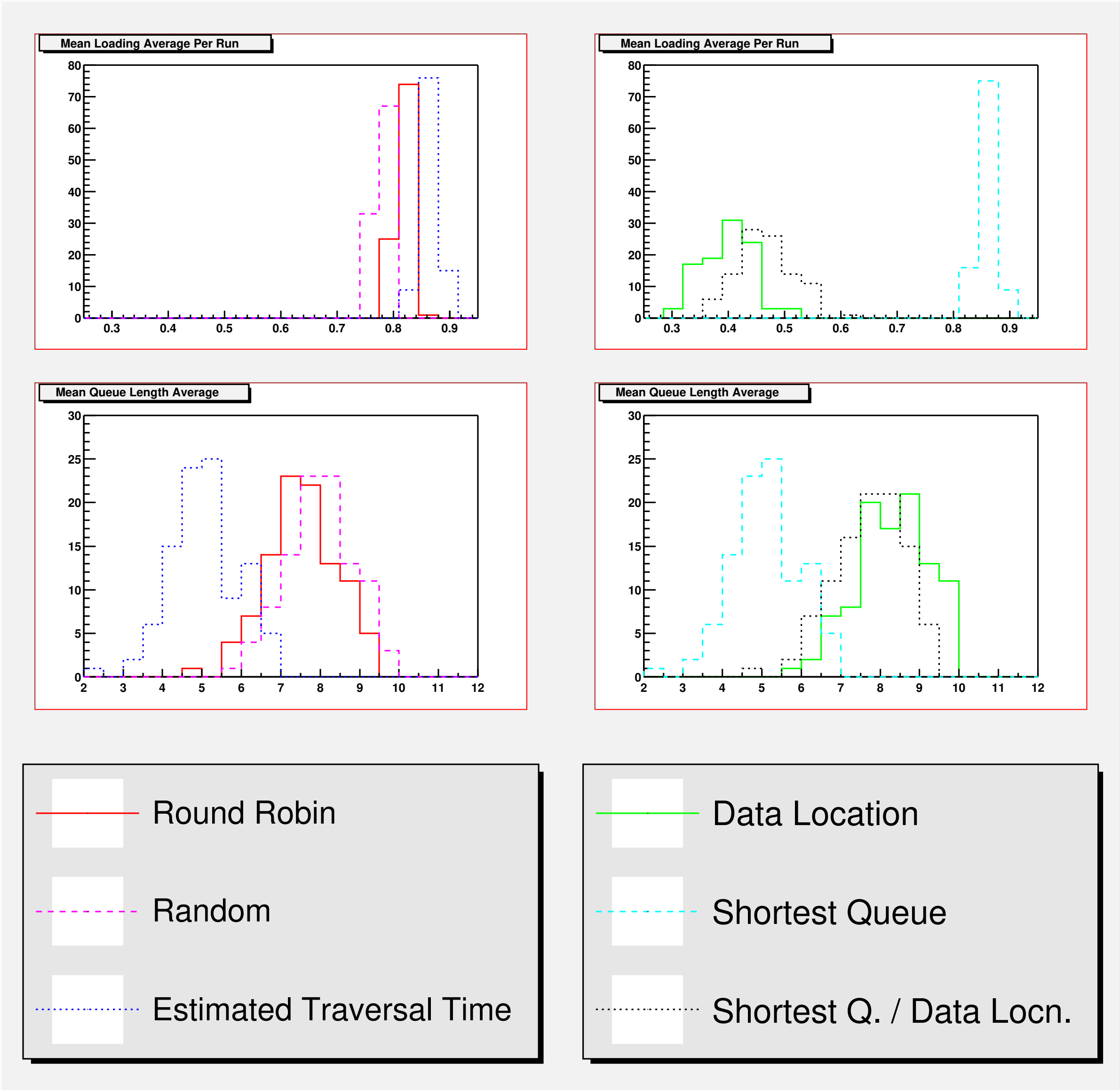

Figure 20: Mean machine loading average (top) and mean queue length (bottom), for six different scheduling algorithms, with a heterogenous Grid and a moderate load of equally sized jobs.

Scheduler	Mean No. Transfers per Run	RMS
Round-Robin	2421	120
Random	2470	116
Estimated Traversal Time	1904	119
Data Location	419	48
Min. Queue Size	1907	116
Data Lcn. / Queue Size	517	54

Table 17: Number of file transfers in a run of the simulation, with a heterogenous Grid and a moderate load of equally sized jobs.

Scheduler	Mean Last Completion Time / 10^5 s	RMS
Round-Robin	3.71	0.20
Random	4.48	0.15
Estimated Traversal Time	2.70	0.19
Data Location	3.73	0.23
Min. Queue Size	2.74	0.19
Data Lcn. / Queue Size	3.54	0.21

Table 18: Last completion time of the last-completed job in a run of the simulation, with a heterogenous Grid and a moderate load of equally sized jobs.

Scheduler	Mean Traversal Time / 10^5 s	RMS
Round-Robin	1.19	0.70
Random	1.32	0.83
Estimated Traversal Time	0.82	0.35
Data Location	1.34	0.75
Min. Queue Size	0.82	0.36
Data Lcn. / Queue Size	1.22	0.69

Table 19: Traversal times of jobs, with a heterogenous Grid and a moderate load of equally sized jobs.

Scheduler	**Mean Loading Avg. per Run**	**RMS**
Round-Robin	0.82	0.014
Random	0.78	0.015
Estimated Traversal Time	0.87	0.015
Data Location	0.40	0.044
Min. Queue Size	0.86	0.016
Data Lcn. / Queue Size	0.47	0.049

Table 20: Mean value of the averaged machine load average over a run of the simulation, with a heterogenous Grid and a moderate load of equally sized jobs.

Scheduler	**Mean Queue Length**	**RMS**
Round-Robin	7.5	0.9
Random	8.0	0.8
Estimated Traversal Time	5.1	0.8
Data Location	8.3	0.9
Min. Queue Size	5.1	0.8
Data Lcn. / Queue Size	7.7	0.9

Table 21: Mean length of queue over a run of the simulation, with a heterogenous Grid and a moderate load of equally sized jobs.

11.2. Variable Job Sizes

The six scheduling algorithms were again tested on the heterogenous Grid, but with a moderate load of jobs with Gaussian distributed CPU requirements, as described in section 9.3.

It was expected that this experiment should relate to the heterogenous Grid / equally sized jobs experiment in the same way that the corresponding experiments with a homogenous Grid topology did. The results are shown in fig. 21 and fig. 22.

This correspondence seems to bear out well, as shown in tables 22-25. Again, the peaked structure disappears with the variation in job sizes breaking the symmetry. The performance of most of the schedulers degrades to a certain extent, as balancing the load of differing jobs presents a more difficult task, but their relative performances stay approximately the same.

The major difference is, again, the performance of the estimated traversal time algorithm. As was seen with the homogenous Grid, the variable job sizes render the estimations of traversal time ineffective (and worse than making a random decision). In this scenario, the minimum queue length algorithm is by some way the most effective. Its use of resources and job processing times are more favourable than its rivals, and it even performs reasonably well in terms of network use (due to its effective utilisation of Site 0 with its larger cache and more permanently hosted files, as described in the previous section).

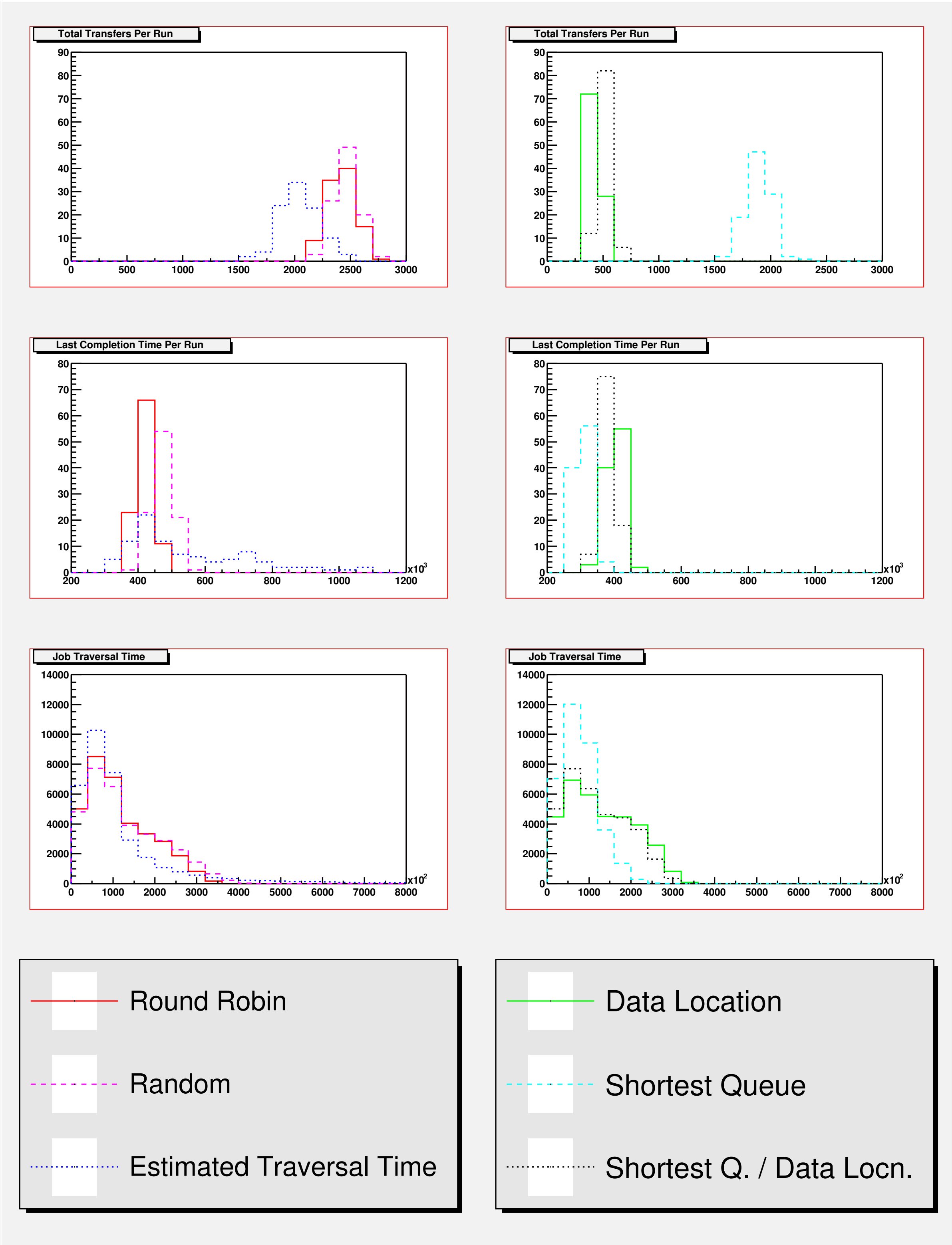

Figure 21: Total transfers per run (top), completion time of last-completed job (middle), and job traversal times (bottom), for six different scheduling algorithms, with a heterogenous Grid and a moderate load of variable sized jobs.

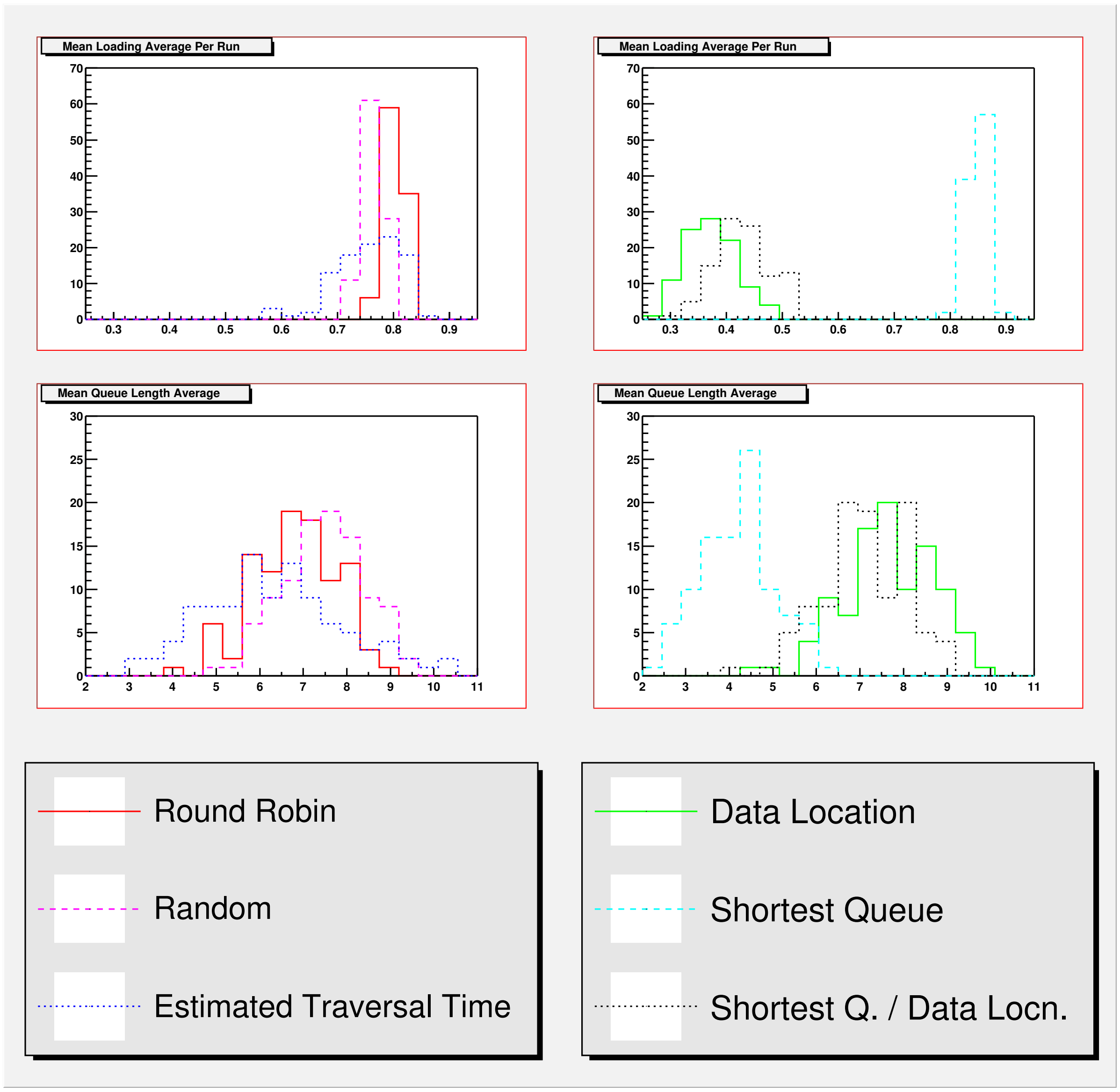

Figure 22: Mean machine loading average (top) and mean queue length (bottom), for six different scheduling algorithms, with a heterogenous Grid and a moderate load of variable sized jobs.

Scheduler	Mean No. Transfers per Run	RMS
Round-Robin	2421	120
Random	2471	116
Estimated Traversal Time	2050	171
Data Location	419	48
Min. Queue Size	1894	122
Data Lcn. / Queue Size	515	53

Table 22: Number of file transfers in a run of the simulation, with a heterogenous Grid and a moderate load of variable sized jobs.

Scheduler	Mean Last Completion Time / 10^5 s	RMS
Round-Robin	4.20	0.25
Random	4.74	0.33
Estimated Traversal Time	4.42	0.67
Data Location	4.01	0.26
Min. Queue Size	3.07	0.25
Data Lcn. / Queue Size	3.81	0.24

Table 23: Last completion time of the last-completed job in a run of the simulation, with a heterogenous Grid and a moderate load of variable sized jobs.

Scheduler	Mean Traversal Time / 10^5 s	RMS
Round-Robin	1.15	0.76
Random	1.28	0.88
Estimated Traversal Time	1.01	0.86
Data Location	1.28	0.78
Min. Queue Size	0.77	0.43
Data Lcn. / Queue Size	1.17	0.72

Table 24: Traversal times of jobs, with a heterogenous Grid and a moderate load of variable sized jobs.

Scheduler	Mean Loading Avg. per Run	RMS
Round-Robin	0.80	0.015
Random	0.76	0.018
Estimated Traversal Time	0.75	0.058
Data Location	0.37	0.045
Min. Queue Size	0.85	0.017
Data Lcn. / Queue Size	0.43	0.048

Table 25: Mean value of the averaged machine load average over a run of the simulation, with a heterogenous Grid and a moderate load of variable sized jobs.

Scheduler	Mean Queue Length	RMS
Round-Robin	6.8	0.9
Random	7.4	0.9
Estimated Traversal Time	6.4	1.6
Data Location	7.7	1.04
Min. Queue Size	4.2	0.8
Data Lcn. / Queue Size	7.11	0.98

Table 26: Mean length of queue over a run of the simulation, with a heterogenous Grid and a moderate load of variable sized jobs.

12. Conclusions

A sophisticated simulation of the European Data Grid has been constructed using the Java-based concurrent design and modelling software Ptolemy II. It consists of an object-oriented representation of an EDG setup, with objects representing the entities involved, including the Resource Broker, Replica Catalog, User Interface, Compute and Storage Elements, and worker machines, as well as the messages and data files passed between them.

The EDG entities themselves have been somewhat simplified, with the emphasis placed more upon their interactions and the flow of jobs and data around the Grid as it runs. The network is represented by a simple Backbone object with no internal structure, as a reasonable first order simulation of a Grid topology.

With this simulated Grid, different job scheduling strategies were tested against each other, as well as against simple round-robin and random schedulers.

An algorithm which estimates the time that jobs will take to be processed (the EDG default scheduler) performed well in scenarios where the jobs had similar demands on CPU resources, but performed significantly less well when these requirements varied. It is suggested that this algorithm would give good results when the lifetime of a job is more dominated by the time spent in the queue than the time it spends running.

An algorithm which sends the job to the largest available concentration of required data gave less impressive results in terms of the time taken to process jobs, but did significantly reduce the level of network use. Its performance was still acceptable with a homogenous Grid topology, but in a heterogenous Grid with resources and data not evenly distributed its performance was very poor (although the reduction in network use was even more pronounced).

Sending jobs to the site with the shortest queue was the most consistent scheduling strategy, giving good results with heterogenous and homogenous topologies, and similar and different CPU requirements for jobs. Although this algorithm was unable to make decisions based upon the requirements of the job it was currently scheduling, it made sensible use of dynamic feedback from the member sites, responding to whether they were under- or over-loaded.

Combining these last two algorithms gave variable results. This strategy consisted of producing a short list of sites with the largest concentration of data, and then selecting from that based upon queue length. With the homogenous Grid, this approach did well, as the job processing times were comparable with the queue-length-only approach, and network use was also reduced. However when it was applied to the heterogenous topology it ran into the same problems as the purely data location based algorithm, albeit with slightly better results.

These results suggest that an intelligent strategy for scheduling can make a significant difference to the performance of a Grid, particularly if its topology is a heterogenous one with its resources unevenly distributed. A poor choice can lead to machines lying idle even when the workload elsewhere is greater than resources can handle.

Combining different strategies to improve scheduling decisions can produce good results, but a bad combination or even combining complementary strategies inappropriately can degrade performances significantly.

Some factors such as a more complex network structure, and variable machine performances have not been included in this work. These are likely to have some effect on the performance of the algorithms tested here. However with the proviso of the simplifications involved in the structure of the simulation, its performance seems to be a convincing representation of a functioning EDG setup, and the relative efficiency of these algorithms has been effectively tested.

A. Future Plans

The work that has been described in this report shows that the PTII simulation of the European Data Grid structure behaves as would be expected, and despite its simplifications can be used to investigate the effects on Grid performance of adjusting certain parameters of its running. However, there is much more work to be done, both in improving the sophistication of representations of the various components, and in carrying out experiments using the simulation.

A.1. Contacts

Closer contact with the people setting up the EDG will be important, in order to improve the accuracy of the simulation. The software is constantly being updated and improved, so it is vital to keep track of these changes and represent them in the simulation.

There are other people working on simulations of the EDG. One such project is Optor, based at Glasgow university, which started out with a focus on data transfer around the Grid, but is now expanding to simulate the whole EDG setup, much as this project aims to. There are also a few other simulation projects being carried out in other parts of the EU.

Perhaps the most important group for the purposes of this work are the team behind Work Package 6. This group is responsible for the testbeds, and its members are thus the people with the most accurate knowledge of what software is currently being implemented, and what they are working to wards. The GridPP WP6 team are currently setting up an EDG-based structure of which UCL is a part, so correspondence with this team could be useful.

A.2. Improved Realism

There are many ways in which this simulation has been simplified, described in the main body of this report. Many of these simplifications are justifiable ones, but there are some that are ideally only temporary measures, and the simulation would become more effective if they were properly developed.

A.2.1. Job Distribution

It is important to develop a more realistic pattern of job generation than the one currently used. The current random distribution is unlikely to be very accurate. A Grid's load is likely to have peaks and troughs in its job load at different times of day. It is likely that a surge in job submission might occur if, say, a new set of data became available that was of interest to a large number of people.

The jobs themselves should be more realistic too, as the simulation is currently using rough estimates of job sizes. A variety of demands might be placed on the system by a job, from large memory requirements to heavy I/O procedures.

Currently the virtual jobs cannot produce output files. Monte Carlo jobs should be implemented, which produce simulated data files comparable in size to the data files already present on the SEs of the Grid. These output files would have to be assigned to an SE, copied over to it, and then registered with the Replica Catalog as a new permanent file. This file could even be used by a job later on in this run of the simulation.

A.2.2. Network Structure

The network structure is one of the biggest simplifications in this work. There is no concept of network topology, so the problems associated with bottlenecks and failures in the network cannot be represented. A network simulation is a large piece of work in itself, and thus outside the scope of this project. But it might be possible for a compromise to be reached.

At the moment the network is represented by a single Actor object called the Backbone. The various sites in the virtual Grid are connected to it, with a local NIC object handling all of the complications of data file transfer, and the Backbone merely passing messages around as it is told to.

It may be feasible instead to have several interconnecting network objects, each with their own bitrate attributes. The connections between them could represent various parts of a network. For instance, the connection between two of these objects could represent a large capacity network backbone running the length of a country. Connected to one of these could be a third object, and this connection could be a much smaller one, perhaps representing a low capacity connection to a small institute.

Message passing would not be problematic - when a network object receives a message, it simply passes broadcasts it in all directions, and all but the intended destination would reject the message. This would then occur again at the next junction, and so on until the message reached its destination.

A.2.3. Multiple Users / Competing Algorithms

Currently the jobs are generated by a single User Interface Actor, connected directly to the Backbone, which then passes it on to the appropriate node, i.e. the one with the Resource Broker. In reality the users will be at the physical nodes in the Grid, and their jobs will be submitted from there. Modification to the Java code would allow a User Interface (or even multiple UIs) to be placed at each node, allowing jobs to be submitted from several different sources.

This opens up new possibilities in the distribution of jobs. Each user can represent their preferences as to how their jobs are scheduled, in the Rank attribute in their JDL / job ClassAd. Different combinations of scheduling preferences may be complementary, or may clash with each other and degrade the performance of the Grid.

A.2.4. Improved Storage Element

Currently the Storage Element begins each run with a set of permanently hosted data files, and an empty cache in which to store temporary replicas. This causes problems with any scheduling algorithm selecting resources based on required data files. The current algorithms only look at permanent files, because the inclusion of replicas in the decision causes a feedback loop at the beginning of the run when the cache is empty.

Once the first few jobs have been assigned, and data copied over for their use, some sites will have slightly more data files resident than others. Since a job's required files are randomly generated, there is a slightly higher probability that these sites with more files locally will be chosen by the scheduler. The job will then be submitted to that site, and files will duly be copied over for it, causing this site to have an even greater proportion of jobs, and so on. Eventually the scheduler ends up submitting all jobs to this site.

Removing the need for this distinction between permanent and replica files, at least from the user / job's perspective is important, as they should only really need to know whether a file is present locally or not. Filling the cache up with temporary files will remove these end effects, and allow the SE to behave more realistically.

In line with a real SE, different types of data access should also be implemented. Some SEs will allow remote data access only, while others would allow users to read from it as a locally mounted disk.

A.2.5. Improved Compute Element

The Compute Element has scheduling decisions of its own to make, as once it is passed a job it must decide which of the processors it is responsible for will receive the new task. As the simulated CEs stand, they simply keep excess jobs in the queue, and when a machine completes a job and becomes available, the next job is assigned to it.

A real CE might have any of the conventional queueing tools in place, able to make more efficient decisions than this. An idle job (e.g. a job unable to run it they waits for the next data file to arrive) could be placed back in the queue and another job could run in its place. Jobs could be prioritised, with a high priority job ousting a previously running job from its machine and running in its place. These technologies should be implemented in the simulated CE.

The ClassAd language is extensible, and allows any property of a resource to be advertised as its owner wishes. There are many more properties available in the standardised EDG resource ClassAd structure which are not currently implemented in this simulation. Some of them allow the CE to have more control over the load on the site, for instance a statement of the maximum number of running and queued jobs that will be accepted. This particular case would allow some regulation of the Grid's total capacity, whereas currently the virtual Grid will keep accepting jobs without rejecting any.

Other characteristics would be useful in the Ranking phase of ClassAd matching, for instance a CPU benchmark value allowing the user to state a preference for sites with faster machines.

A.2.6. Component Failure

A Grid is such a large and complex entity that failure of some of its components is to be expected, and in fact is likely to be the norm. The components of this simulation are all infallible, and it could be interesting to see what might happen were some of them to fail occasionally.

If resources were to fail, the Resource Broker would have to be sure not to send new jobs to the defunct site until it came back on line. Greater difficulties would be associated with rescheduling the jobs that were running at that resource, and determining whether any results of the job's running so far could be rescued, or if the job would have to be re-run from scratch. If it were a Storage Element that had failed, acquiring the necessary data for a job might become more difficult, or even impossible.

More important components could be made unreliable. If the Resource Broker failed, jobs could not be submitted, unless a backup RB was available. A faulty RC would make it impossible to find the required data for a job, unless the user submitted a list of specific PFNs. If the Information Services were slow or not functioning, the RB could be scheduling jobs based on redundant information, leading to inefficient scheduling.

A.3. Future Experiments

There are interesting experiments still to be carried out with the simulation, even as it stands currently. Implementing some of the improvements described above will allow new experiments, usefully answering questions about the setup of the EDG without the need for testing with a real Grid.

A.3.1. Budgeting

If money becomes available to upgrade sites in a Grid, it is not obvious what would be the best way to spend this money. New machines could be bought, or the available storage space could be upgraded. The network connection at a site could be upgraded, to facilitate transfer of files. In a homogenous Grid, it might be unclear where the money should be spent, and which resources should be replaced or upgraded. The simulation could be used to test different configurations, and determine how best to improve the functioning of the Grid.

Conversely, a certain desired level of performance could be stated, in terms of job performances or total throughput. The minimum budget necessary to reach this level of improvement could be determined, by testing different configurations and upgrades.

A.3.2. Competing / Complementary Algorithms - Greed vs. Cooperation

As described above, implementing multiple User Interfaces with different scheduling preferences allows interesting possibilities. Even if it is possible to find an 'ideal' individual scheduling algorithm, overall performance might be improved by a clever combination of algorithms.

One user might request that their jobs be run on the fastest possible machines. While this might not lead to efficient use of the Grid as a whole, this user will be happy, as his/her jobs will be run quickly.

A second user could declare the same preferences, which would lead to a conflict, as both users' jobs would be competing for the same resources. The second user might lose out and end up behind in CE queues, if the other got in there first.

However the second user could instead request the opposite, and actually have their jobs sent to the less well resourced CEs. This might not produce the best performance, but if these CEs were unpopular, this user's jobs would be able to start running straight away, and might still return their jobs quickly.

This is a simplistic example, but different combinations of schedulers could be tested to see what effect they had on individual users' jobs, and the throughput of jobs in the Grid as a whole.

One could also ask whether the users, who are bound to be at least a little selfish, should be in complete control of the scheduling in the Grid. The EDG Resource Broker as it stands will only interpret the requests of the users, and base its decisions upon that. Perhaps the RB could be given some power to override the users, for instance in cases where user requests lead to only a certain subset of Grid resources being used. Differing degrees of RB intervention could be introduced, to investigate the trade-off between user satisfaction, and load sharing across the Grid.

A.3.3. Economic Model

So far in this work there has been no consideration of the relationship between resource owners and users, who is allowed to use the resources, and how much. An accounting system has been suggested, in order to regulate this.

A system of credit must be devised, whether it is real-world money, or notional Grid credits. The users must give up some of their credits to use a resource, and resource owners receive credits for making their machines available. Thus no one would be able to unfairly swamp the Grid with jobs, as they would not have enough credits to do so.

This ties in to the idea of prioritisation. If a user has a job that must be run urgently, they can pay extra credits for their job to be prioritised over other users' jobs. If they are not concerned with a rapid return of results, they pay less for their job submission.

The effect of this on the running of the EDG, and whether it would complement, conflict with, or effectively replace the current system of job scheduling could be investigated.

A.3.4. Virtual Organisations

In a Grid for use by high energy physicists across Europe, people working on many different experiments may be using the same set of resources, as a given institute will have members of multiple HEP projects located at the same site. In Grid terminology, this means that the institute's resources belong to several Virtual Organisations.

This leads to questions about the structure of the EDG components. Do the different VOs submit all of their jobs to the same Resource Broker? The experiments are likely to have overlapping sets of resources, rather than all VOs having access to all resources. Should each VO have its own RB? This becomes a little like the question of competing or complementary scheduling policies. If only one RB is used for all VOs, it will have to factor VO membership into its decisions.

The question of which of these possibilities leads to the best use of resources, from the point of view of the users, as well as that of load balancing across the Grid. These different structures can be implemented, and their results compared.

A.3.5. Comparison with SAM

Currently being implemented at Fermilab is a Grid structure called Sequential Access to Metadata, or SAM. This shares some characteristics with the EDG, although it differs in some aspects such as its storage and handling of files. Other aspects of its structure are yet to be finalised, and may differ further from the EDG.

It might be possible to compare and contrast the performance of both structures, if the SAM structure does not fall too far from that of the EDG, and does not require a prohibitive amount of work to develop the code. Their relative strengths and weaknesses could be investigated, hopefully to the benefit of both projects.

References

[1] I. Foster and C. Kesselman, *The Grid; Blueprint for a New Computing Infrastructure*, Morgan Kauffmann (1999).

[2] J. Nieploca and R. Harrison, *Shared memory programming on the I-WAY*, Proc. Fifth IEEE Symp. on High Performance Distributed Computing p. 432-441, Los Alamitos, CA: IEEE Computer Society Press (1996).

[3] M. Norman et al., *Galaxies collide on the I-WAY: An example of heterogeneous wide-area collaborative supercomputing*, Int'l J. Supercomputer Applications, **10(2)**, 131-140 (1996).

[4] J. Czyzyk, M. P. Mesnier and JJ. More, *The Network-Enabled Optimization System (NEOS) Server*, Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, IL (1996).

[5] H. Casanova and J. Dongarra, *Netsolve: A network server forsolving computational science problems*, Tech. report CS-95-313, University of Tennessee (1995).

[6] G. H. Wheless et al, *Virtual Chesapeake Bay: Interacting with a coupled physical/biological model*, IEEE Computer Graphics and Applications, **16(4)**, 42-43 (1996).

[7] M. Roussos et al., *NICE: Combining constructionism, narrative, and collaborating in a virtual learning environment*, Computer Graphics, **31(3)**, 62-63 (1997).

[8] C. Potter et al., *EVAC: A virtual environment for control of remote imaging instrumentation*, IEEE Computer Graphics and Applications, p. 62-66 (1996).

[9] L. Gravano et al., *STARTS: Stanford proposal for internet meta-searching*, in Conf. on Management of Data (1997).

[10] J. Pfaendtner et al., *Documentation of the Goddard Earth Observation System (GEOS), Data Assimilation System - Version 1*, http://dao.gsfc.nasa.gov/

[11] A. Grimshaw et al., *Legion: An Operating System for Wide Area Computing*, IEEE Computer, **32(5)**, 29-37 (1999).

[12] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, Int'l J. Supercomputer Applications, **11(2)**, 115-128 (1997).

[13] CCITT Recommendation X.509: The directory authentication framework (1988).

[14] M. Litzkow, M. Livny and M. W. Mutka, *Condor - a hunter of idle workstations*, Proc. Eighth Int'l Conf. of Distributed Computing Systems, p. 104-111 (1988).

[15] D. Abramson et al., *Nimrod: A tool for performing parameterised simulations using distributed workstations*, in Proc. Fourth IEEE Symp. on High Performance Distributed Computing, Los Alamitos: IEEE Computer Society Press (1995).

[16] Particle Physics Data Grid, http://www.ppdg.net/

[17] GriPhyN (Grid Physics Network), http://www.griphyn.org/

[18] iVDGL (International Virtual Data Grid Laboratory, http://www/.idvgl.org/

[19] The LHC Computing Grid Project, http://lcg.web.cern.ch/LCG/

[20] *DataTAG: Research and Technological Development for a Transatlantic Grid, Project Description*, http://datatag.web.cern.ch/datatag/project.html

[21] The MONARC Architecture Group, *Regional Centres for LHC Computing*, http://monarc.web.cern.ch/MONARC/docs/monarc_docs/1999-03.html

[22] The European DataGrid Project, *An Introduction to DataGrid*, http://web.datagrid.cnr.it/LearnMore.jsp