Communicating Process Architectures 2005
Jan Broenink, Herman Roebbers, Johan Sunter, Peter Welch, and David Wood (Eds.)
IOS Press, 2005

The Architecture of the Minimum intrusion Grid, MiG

Brian Vinter
University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark

Abstract. This paper introduces the philosophy behind a new Grid model, the Minimum intrusion Grid, MiG. The idea behind MiG is to introduce a 'fat' Grid infrastructure which will allow much 'slimmer' Grid installations on both the user and resource side. This paper presents the ideas of MiG, some initial designs and finally a status report of the implementation.

1. Introduction

Grid computing is just around the top of the hype-curve, and while large demonstrations of Grid middleware exist, including Globus toolkit[8] and NorduGrid ARC[9], the tendency in Grid middleware these days is towards a less powerful model, Grid services, than what was available previously. This reduction in sophistication is driven by a desire to provide more stable and manageable Grid systems. While striving for stability and manageability is obviously right, doing so at the cost of features and flexibility is not so obviously correct.

The Minimum intrusion Grid, MiG, is an attempt to design a new platform for Grid computing which is driven by a stand-alone approach to Grid, rather than integration with existing systems. The goal of the MiG project is to provide a Grid infrastructure where the requirements on users and resources alike, to join Grid, are as small as possible – thus the minimum intrusion part. While striving for minimum intrusion, MiG will still seek to provide a feature rich and dependable Grid solution.

2. Grid Middleware

The driving idea behind the Minimum intrusion Grid project is to develop a Grid[7] middleware that allows users and resources to install and maintain a minimum amount of software to join the Grid. MiG will seek to allow very dynamic scheduling and scale to a vast number of processors. As such MiG will close the gap between the existing Grid systems and popular "Screen Saver Science" systems, like SETI@Home.

2.1.1 Philosophy behind MiG

"The Minimum intrusion Grid", this really is the philosophy - we want to develop a Grid middleware that makes as few requirements as possible. The working idea is to ensure that a user needs only a signed x509 certificate, trusted by Grid, and a web-browser capable of secure HTTP, HTTPS[10]. A resource on the other hand must also hold a trusted x509 certificate and in addition create a user – the Grid user – who can use secure shell, ssh, to enter the resource and once logged on can open HTTPS connections to the outside. The requirements then become:

	User	Resource
Must have certificate	Yes	Yes
Must have outbound HTTP	Yes	Yes
Must have inbound SSH	No	Yes

Table 1. Requirements for using MiG

2.2 What's wrong with the existing Grid systems?

While there are many Grid middleware systems available most of them are based on, or descendents of, the Globus toolkit. Thus the description below addresses what the author believe to be shortcomings in the Globus toolkit, and not all issues may be relevant to all Grid systems.

2.2.1 Single point of failure

Contrary to popular claim, all existing Grid middlewares hold a central component that, if it fails, requires the user to manually choose an alternative. While the single point of failure may not truly be a single point, but comply with some level of redundancy, none of the components scale with the size of the Grid.

2.2.2 Lack of scheduling

All existing systems perform a job-to-resource mapping. However, an actual scheduling with a metric of success is not available. Work is underway in this in the community scheduler[16] but for this scheduler to work, the resources need to be exclusively signed over to Grid, i.e. a machine can not be accessed both through Grid and a local submission system.

2.2.3 Poor scalability

The time taken to perform the job-to-resource mapping in the current systems scales linearly with the number of sites that are connected. This is already proving to be a problem in NorduGrid, which is one of the largest known Grids, though only 36 sites are connected. Imagining tens of thousands of connected sites is not likely. In the Grid service model scalability issues are more or less eliminated by absence of a single system view from a user perspective.

2.2.4 No means of implementing privacy

The job submission API at the users machine communicates directly with all the potential sites, thus all sites know the full identity of all jobs on the Grid.

2.2.5 No means of utilizing 'cycle-scavenging'

Cycle-scavenging, or Screen Saver Science, utilizes spare CPU cycles when a machine is otherwise idle. This requires an estimate on how long the machine will be available and all existing systems just assume that a free resource will be available indefinitely. The model has been partly demonstrated in NorduGrid by connecting a network of workstations running Condor, to NorduGrid, but Grid itself has no means of screen-saver science.

2.2.6 Requires a very large installation on each resource and on the user site

The middleware that must be installed on a resource to run NorduGrid, which is probably the best

of the well known Grid middlewares[1], is more than 367 MB including hundreds of components. All of which must be maintained locally. This means that donating resources to Grid is associated with significant costs for maintenance; this naturally limits the willingness to donate resources.

2.2.7 Firewall dependency

To use the existing middlewares special communication ports to the resource must be opened in any firewall that protects a resource. This is an obvious limitation for growing Grid since many system-administrators are reluctant towards such port-openings. One project that seeks to address this problem is the centralized-gateway-machine project under the Nordic Data Grid Facility[17] that receives jobs and submits them to the actual resource using SSH.

2.2.8 Highly bloated middleware

The existing middleware solutions provide a very large set of functions that are placed on each site, making the software very large and increasing the number of bugs, thus the need for maintenance, significantly.

2.2.9 Complex implementation using multiple languages and packages

The current Grid middlewares have reused a large amount of existing solutions, for data-transfer, authentication, authorization, queuing, etc. These existing solutions are written in various languages and thus the Grid middleware uses more than 6 programming languages and several shell types, in effect raising the cost of maintaining the package further. The many languages and shells also limit portability to other platforms.

3. MiG Design Criteria's

MiG should design and implement a functional Grid system with a minimal interface between the Grid, the users, and the resources. The successful MiG middleware implementation should hold the following properties.

3.1.1 Non-intrusive

Resources and users should be able to join Grid with a minimum of effort and with a minimum software installation. The set of requirements that must be met to join Grid should also be minimal. "Minimal" in this context should be interpreted rigidly, meaning that if any component or functionality in MiG can be removed from the resource or user end, this must be done, even if adding the component at the resource or user end would be easier.

3.1.2 Scalable

MiG should be able to contain tens of thousands, even millions, of resources and users without the size of the system impacts performance. Even individual PCs should be able to join as resources. For a distributed system, such as MiG, to be truly scalable it is necessary that the performance of the system is not reduced as the number of associated computers grows.

[1] NorduGRID is the worlds largest multi-discipline Grid and is frequently used for arguing for new features in other Grid systems.

3.1.3 Autonomous

MiG should be able to perform an update of the Grid without changing the software on the user or resource end. Thus compatibility problems that arise from using different software versions should be eliminated by design. To obtain this feature it is necessary to derive a simple and well defined protocol for interaction with the Grid middleware. Communication within the Grid can be arbitrarily complex though since an autonomous Grid architecture allows the Grid middleware to be upgraded without collaboration from users and resources.

3.1.4 Anonymous

Users and resources should not see the identity of each other if anonymity is desired. This is a highly desirable feature for industrial users that are concerned with revealing their intentions to competing companies. A highly speculative, example could be two pharmaceutical companies A and B. Company A may have spare resources on a computational cluster for genome comparisons, while B may be lacking such resources. In a non-anonymous Grid model, company B will be reluctant to use the resources at company A since A may be able to derive the ideas of B from the comparisons they are making. However, in a Grid that supports anonymous users, A will not know which company is running which comparisons which makes much less valuable. In fact many comparisons will be likely to be part of research projects that map genomes and will thus reveal nothing but information that is already publicly available.

3.1.5 Fault tolerance

Failing machines or processes within the Grid should not stop users or resources from using the Grid. While obvious, the lack of fault tolerance is apparent in most Grid middlewares today. The consequences of lacking fault tolerance range from fatal to annoying. Crashes are fatal when a crashed component effectively stops users from running on Grid, i.e. a hierarchy of Meta Directory Servers.

If a resource that runs users' processes crash it becomes costly for the users that are waiting for the results of the now lost jobs. Finally crashes are merely annoying when a crashed component simply does not reply and thus slows down the users interactions with the Grid because of timeouts.

3.1.6 Firewall compliant

MiG should be able to run on machines behind firewalls, without requiring new ports to be opened in the firewall. While this requirement is quite simple to both motivate and state, actually coping within the restraints of this point may prove highly difficult.

3.1.7 Strong scheduling

MiG should provide real scheduling, not merely job-placement, but it needs to do so without requiring exclusive ownership of the connected resources. Multi-node scheduling should be possible as should user-defined scheduling for dynamic subtasking. In effect MiG should also support meta-computing[2].

[2] Metacomputing is a concept that precedes Grid computing. The purpose of metacomputing is to create a large virtual computer for executing a single application.

3.1.8 Cooperative support

In order to improve the meta-computing qualities, MiG should provide access to shared user-defined data-structures. Through these data-structures a MiG based Grid system can support collaborating applications and thus improve the usability of Grid.

4. The abstract MiG model

The principal idea behind MiG is to provide a Grid system with an overall architecture that mimics a classic, and proven, model – the Client-Server approach. In the Client-Server approach the user sends his or her job to the Grid and receives the result. The resources, on the other hand, send a request and receive a job. After completing the job the resource sends the result to the Grid which can forward the reply to the user.

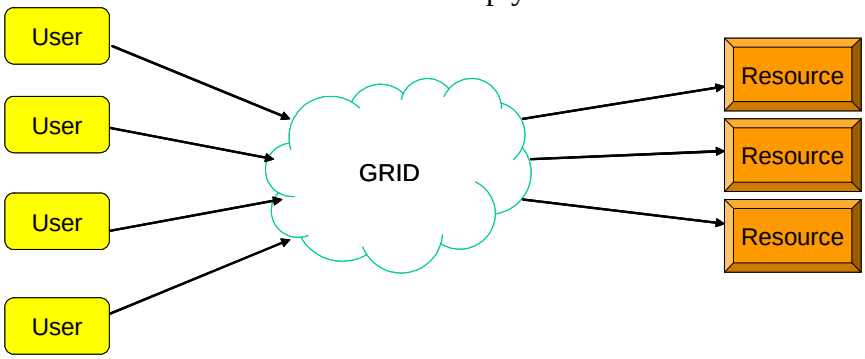

Figure 1. The abstract MiG model

The Grid system should be disjoint from both the users and the resources, thus the Grid appears as a centralized black-box to both users and resources.

This model allow us to remain in full control of the Grid, thus upgrades and trouble shooting can be performed locally within Grid, rather than relying on collaboration from a large number of system-administrators. In addition, moving all the functionality into a physical Grid system, lowers the entry level that is required for both users and resources to join, thus increasing the chances that more users and resources do join the Grid.

In MiG, storage is also an integrated component and users will have their own ‘home directory’ on MiG, which can be easily accessed and referenced directly in job-descriptions so that all issues with storage-elements and replica catalogs is entirely eliminated.

For a user to join, all that is required is an x509 certificate which is signed by a certificate authority that is trusted by MiG. Accessing files, submitting jobs and retrieving results can the all be done through a web-browser that supports certificate based HTTPS. As a result the user need not install any software to access Grid and if the certificate is carried on a personal storage device, e.g. a USB key, a user can access Grid from any internet enabled machine.

The requirements for resources to join MiG should also be an x509 certificate, but in addition the resource must create a Grid account in which Grid jobs are run. Initially MiG requires that this user can SSH into the account, but alternatives to this model will be investigated.

4.1 The simple MiG model

In a simple version of the MiG model there's only a single node acting as the Grid. Clients and resources then communicate indirectly through that Grid-node. The interface between the user and Grid should be as simple as possible. The exact protocol remains a topic for

investigation but, if possible, it will be desirable to use only the HTTP[10] protocol or a similar widely used, and trusted, protocol. Towards the resources the protocol should be equally simple, but in this case, as we also desire that no dedicated Grid service is running on the resource, one obvious possibility is to use the widely supported SSH[11] protocol.

When submitting a job, the user sends it to the Grid machine which stores the job in a queue. At some point a resource requests a job and the scheduler chooses a job to match the resources that are offered. Once the job is completed the results are sent back to MiG. The user is informed that the job has completed and can now access MiG and retrieve the results.

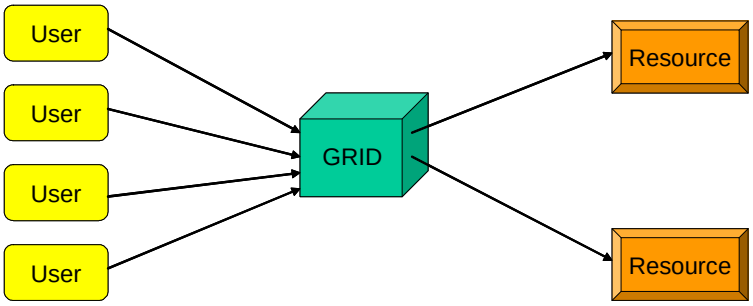

Figure 2. The simple MiG model

4.1.1 Considering the simple model

The simple model of course, is quite error-prone as the single Grid machine becomes both a single point of failure and a bottleneck which is not acceptable. The obvious solution is to add more Grid machines which can act a backup for each other.

4.2 The full MiG model

The obvious flaw in using the client-server model is that achieving robustness is inherently hard in a centralized server system where potential faults include:

- Crashed processes
- Crashed computers
- Segmented networks
- Scalability issues

To correctly function in the presence of errors, including the above, error redundancy is needed. The desired level of redundancy is a subject to further investigations, but should probably be made dynamic to map the requirements of different systems. To address the performance issues Grid itself must be distributed so that users can contact a local Grid server. Thus workload will be distributed through the physical distribution of users.

Once a job arrives at a Grid server the server must ensure that the job is “deposited” at a number of other servers, according to the current replication rate. The user should not receive an acknowledgement of submission before the job has been correctly received and stored at the required number of servers.

Once a resource has completed a job the resource is expected to deliver the result. If, however, the client has not provided a location for placing the result, the resource can still insist on uploading the results. To facilitate this, the Grid should also host storage to hold results and user input-files, if a resource cannot be allocated at the time the client submits his job.

To facilitate payment for resources and storage a banking system should be implemented. To allow inter-organization resource exchange, the banking system should support multiple banks. Dynamic price-negotiation for the execution of a job is a very attractive component that is currently a research topic. Supporting price-negotiations in a

system such as MiG where no central knowledge is available is an unsolved problem that must be addressed in the project. Likewise, scheduling in a system with no central coordination is very hard.

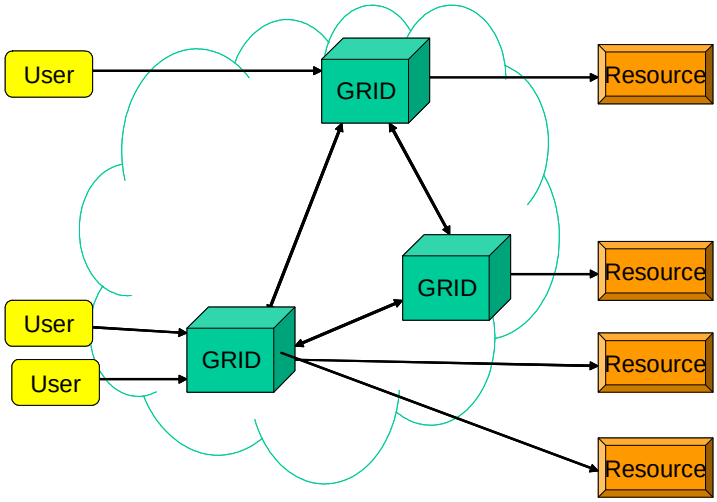

Figure 3. The full MiG model

4.2.1 Considering the full model

One topic for further investigations is: how do we schedule on multiple Grid servers? In principle we would prefer complete fairness, so that the order in which jobs are executed is not dependent on where they are submitted, i.e. to which MiG node. Such a full coordination between all nodes in MiG for each job-submission is not realistic since it will limit scalability, thus a model that allows scalability while introducing some level of load-balancing and fairness will have to be invented.

4.3 MiG Components

4.3.1 Storage in MiG

One difficulty that users report when using Grid is file access. Since files that are used by Grid jobs must be explicitly uploaded to a Grid storage element, result files must be downloaded equally explicitly. On the other hand it is a well known fact that the expenses associated with a professional backup strategy often prohibit smaller companies from implementing such programs, and relies on individual users to do the backup - a strategy that naturally results in a large loss of valuables annually. Some interesting statistics include[18]:

- 80% of all data is held on PCs (Source, IDC)
- 70% of companies go out of business after a major data loss (Source, DTI)
- 32% of data loss is due to user error (Source, Gartner Group)
- 10% of laptops are stolen annually (Source, Gartner Group)
- 15% of laptops suffer hardware failure annually (Source, Gartner Group)

By using the Grid, we do not just gain access to a series of computational resources, but also to a large amount of storage. Exploitation of this storage is already known about from peer-to-peer systems, but under “well-ordered" conditions it can be used for true Hierarchal Storage Management, HSM. When working with HSM the individual PC or notebook only has a working copy of the data which is then synchronized with a real dataset located on Grid. By introducing a Grid based HSM system, we can offer solutions to two important issues at one time; firstly Grid jobs can now refer directly to the dataset in the home-catalog

thus eliminating the need for explicit up- and down-loads of files between the PC and Grid. Second, and for many smaller companies much more importantly, we can offer a professionally driven storage-system with professional backup solutions, either conventional backup systems or, more likely, simple replica based backup - the latter is more likely because disks are becoming rapidly more inexpensive and keeping all data in three copies is easily cheaper than a conventional backup-system and the man-power to run it. A Grid based HSM system will allow small companies to outsource the service while medium and large companies can chose to either outsource or implement a Grid HSM in-house. Thus by introducing Grid based HSM, Grid can offer real value to companies that are not limited by computational power and these companies will thus be "Grid integrated" when Grid becomes the de-facto IT infrastructure.

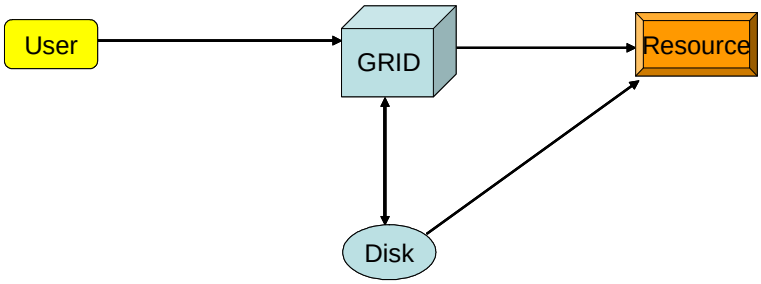

Figure 4. MiG Storage support

4.3.2 Scheduling

Scheduling in Grid is currently done at submission-time and usually a scheduled task is submitted to a system where another level of scheduling takes place. In effect the scheduling of a job provides neither fairness for users nor optimal utilization of the resources that are connected to the Grid, and the current scheduling should probably just be called job-placement. Furthermore, the current model has a built in race-condition since the scheduling inquires all resources and submits to the one with the lowest time-to-execute. If two or more jobs are submitted at the same time they will submit to the same resource, but only one will get the expected timeslot. The MiG model makes scheduling for fairness much simpler as the local scheduling comes before the Grid scheduling in the proposed model.

Scheduling for the best possible resource utilization is much harder and of much more value. The problem becomes one that may be described as: given the arrival of an available resource, and an existing set of waiting jobs, which job is chosen for the newly arrived resource so that the global utilization will be as high as possible?

The above is in the common case where jobs are more frequent than resources, in the rare case that resources are more abundant than jobs, the same problem is valid on the arrival of a job.

When scheduling a job, future arrivals of resources are generally not known, i.e., we are dealing with an *on-line* scheduling problem. On-line scheduling is an active research area, initiated as early as 1966[1] and continued in hundreds of papers, see [2] and [3] for a survey. This problem, however, differs from all these on-line scheduling problems investigated previously in that the resources, not the jobs, arrive over time in the common case. The problem also has some similarity with on-line variable-sized bin packing [4][5][6], but again with a twist that has not been considered before; the bins, not the items to be packed, arrive on-line.

4.3.3 *Security and Secrecy*

In Grid, security is inherently important, and the MiG system must be at least as secure as the existing systems. The simple protocols and minimal software based on the resources make this goal easy to achieve, but still the mechanisms for security must be investigated. Secrecy is much harder and is currently not addressed in Grid. Privacy will mean much towards achieving secrecy but other issues are also interesting topics of research. I.e. if a data file is considered valuable, e.g. a genomic data sequence, how can we hold the contents of that file secret to the owner of the resource? In other words, can MiG provide means of accessing encrypted files without asking the users to add decryption support to his application?

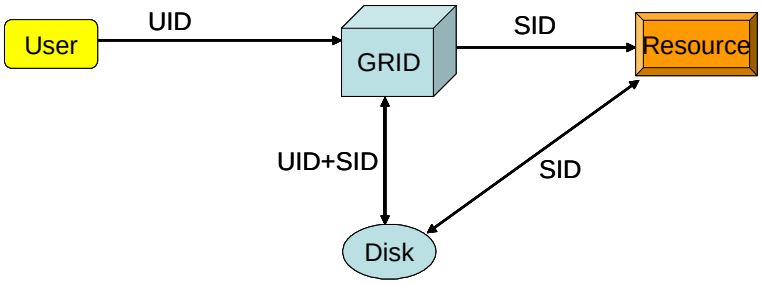

Figure 5. Anonymity and security model

4.3.4 *Fault-tolerance*

In a full Grid system errors can occur at many levels and failures must be tolerated on MiG nodes, resources, network connections and user jobs. Any single instance of these errors must be transparent to the user. More complex errors of course, or combinations of the simple errors, cannot fully be hidden from the users, i.e. if a user is on a network that is segmented from the remaining internet we can do nothing to address this.

Achieving fault tolerance in a system such as MiG is merely a question of never loosing information when a failure occurs, e.g. keeping redundant replicas of all information. Figure 6 shows how a submitted job is replicated when it is submitted.

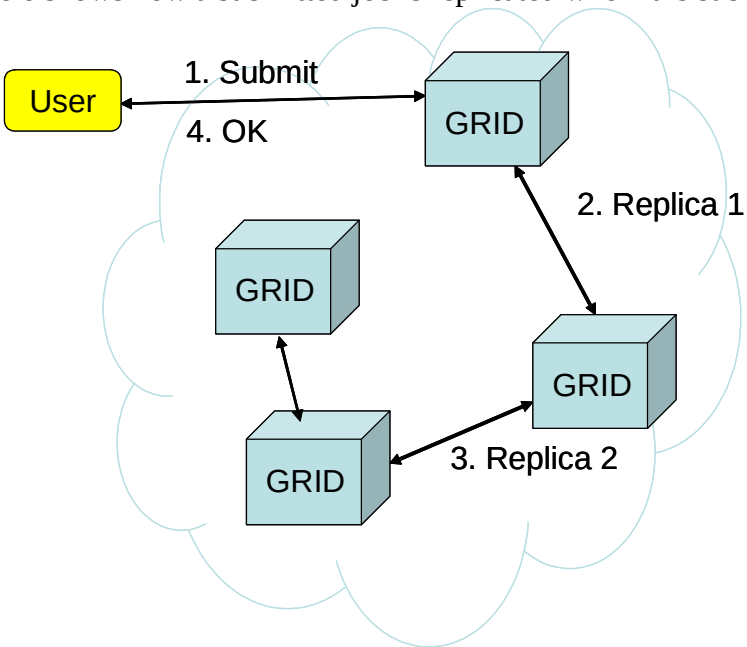

Figure 6. Replicating a new job

Recovering from a failure is then a simple matter of detecting the failure and restoring the required number of replica's as shown in Figure 7 where the number of replicas is three.

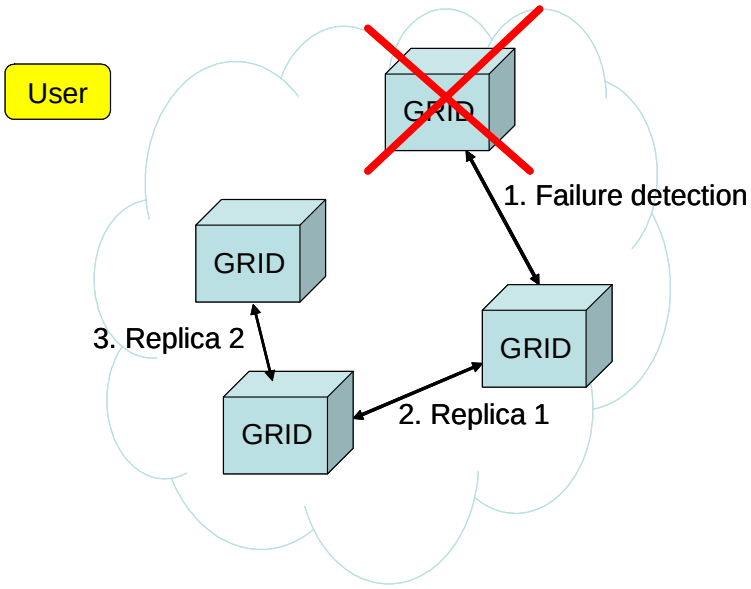

Figure 7. Recovering from a failure

4.3.5 Load balancing and economics

Load balancing in distributed systems is an interesting and well investigated issue[13]. However load balancing for, potentially, millions of resources while maintaining a well defined measure of fairness is still an unsolved issue. However adding economics to the equation actually makes this easier. Since MiG should support a market oriented economy, where the price for executing a job is based on demand and supply, this introduces a simple notion of fairness which is that resources should optimize their income while users should minimize their expenses.

In case there are more jobs than resources, which is the common case, the next job to execute is the job that is willing to pay most for the available resource. In case two or more jobs bid the same for the resource the oldest of the bidders is chosen.

In the rare case that there are more resources offering their services than there are jobs asking for a resource, the next available job is sent to the resource that will sell its resources cheapest. In case more resources bid at the same price, the one that have been waiting the longest wins the bid.

4.3.6 Shared data-structures for MiG

When people with no knowledge of Grid computing are first introduced to Grid, they often mistake it for meta-computing and expect the Grid to behave as one large parallel processor and not a large network of resources. This misunderstanding is quite natural, since such a Grid computing model would be highly desirable for some applications, of course most parallel applications cannot make use of such an unbalanced virtual parallel processor. However, to support the applications that can make use of Grid as a meta-computing system, MiG will provide support for shared data-structures which are hosted on Grid.

4.3.7 Accounting/Price-negotiations

Grid becomes really interesting once users can purchase resources on Grid, thus transforming Grid from a resource sharing tool into a market place. To support this vision, MiG will not only do accounting but also support a job bourse, where the price for a task can be dynamically negotiated between a job and a set of resources. Such dynamic price-

setting is also a known subject, but combining it with load-balancing and fairness in a truly distributed system has not been investigated.

4.3.8 User defined scheduling

An advanced extension of the online-scheduling problem is the subtasking problem, where a job may be divided into many subjobs. If the subtasks have a natural granularity the task is trivial and known solutions exist, including functioning systems, such as SETI@Home. If, on the other hand, a subtask can be selected that solves the largest possible problem on the given resource, the problem becomes very hard and no system provides means for this today.

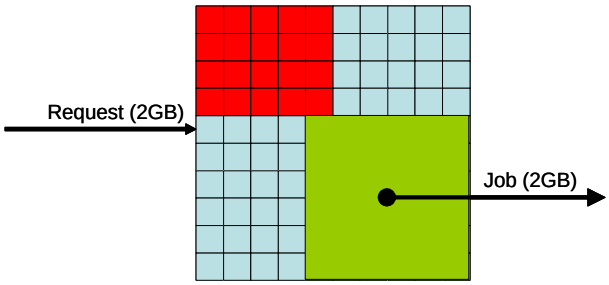

Figure 8. Dynamic sub-scheduling

When comparing with on-line bin packing, this variant of the problem has one further twist to it; the size of an item (a subtask) may depend on which other items are packed in the same bin, since the data needed by different subtasks may overlap.

MiG will seek to develop a model where a job can be accompanied with a function for efficient sub-tasking. The demonstration application for this will be a new version of the Grid BLAST application, which is used in Bio-Science for genome comparisons. The efficiency of BLAST depends on two parameters; input-bandwidth and available memory. We currently developing a dynamic subtasking algorithm that creates subjobs fitted for resources as they become available.

4.3.9 Graphics rendering on Grid

Currently Grid is used exclusively for batch job processing. However for Grid to truly meet the original goal of “computing from a plug in the wall”, graphics and interactivity is needed. In this respect MiG makes things more complex than the existing middlewares since MiG insists on maintaining anonymity, e.g. we insist that a process can render output to a screen-buffer that it cannot know the address of.

The solution to this problem is similar to the storage model. A ‘per-user’ frame-buffer is hosted in the MiG infrastructure, and resources can render to this, anonymous, region. Users on the other hand can choose to import this buffer into their own frame-buffer and thus observe the output from their processes without the hosts of these processes knowing the identity of the receiver. The approach for anonymous rendering in MiG is sketched in Figure 9.

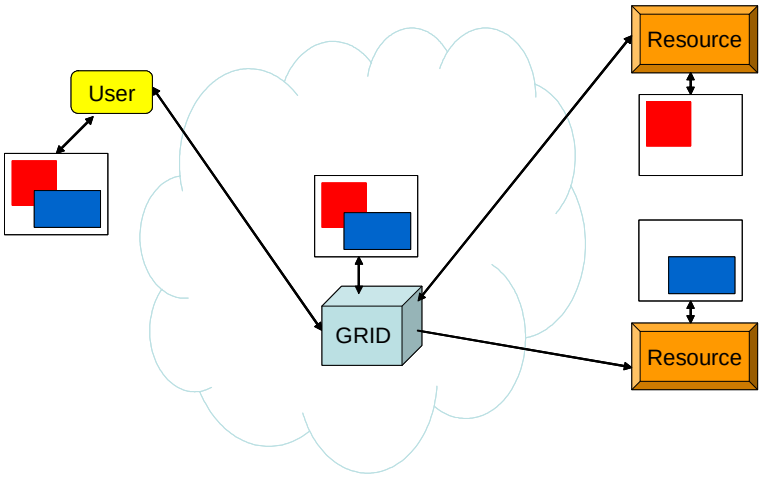

Figure 9. Anonymous graphics rendering in MiG

4.4 Status

At the time of writing MiG is fully functional from a usage perspective, all the features described in 4.1, the simple model, further details on the implementation can be found in [14]. The storage model described in section 4.3.1 is also fully functional and described in some detail in [15].

5. Conclusions

The purpose of this paper is to motivate the work on a new Grid middleware, the Minimum intrusion Grid, MiG. MiG is motivated in a set of claimed weaknesses of the existing Grid middleware distributions, and a desire to develop a model for Grid computing that is truly minimum intrusion.

The proposed model will provide all the features known in today's Grid systems, and a few more, while lowering the requirements for a user to simply having an X.509 certificate, and for a resource to have a certificate and create a Grid-user who can access the resource through SSH.

While MiG is still in its very initial stage, users can already submit jobs and retrieve their results, while maintaining complete anonymity from the resource that executes the job.

References

[1] R. L. Graham, Bounds for Certain Multiprocessing Anomalies, Bell Systems Technical Journal, vol 45, 1563--1581, 1966

[2] Y. Azar, On-Line Load Balancing, Online Algorithms: The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[3] J. Sgall, On-Line Scheduling, Online Algorithms: The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[4] J. Csirik, An On-Line Algorithm for Variable-Sized Bin Packing, Acta Informatica, 26, pp 697--709, 1989.

[5] J. Csirik and G. Woeginger, On-Line Packing and Covering Problems, Online Algorithms: The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[6] L. Epstein and L. M. Favrholdt, On-Line Maximizing the Number of Items Packed in Variable-Sized Bins, Eighth Annual International Computing and Combinatorics Conference (to appear), 2002

[7] I. Foster. The Grid: A New Infrastructure for 21st Century Science. Physics Today, 55(2):42-47, 2002.

[8] I. Foster, C. Kesselman. The Globus Project: A Status Report. Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18, 1998.
[9] P. Eerola et al. "Building a Production Grid in Scandinavia". IEEE Internet Computing, 2003, vol.7, issue 4, pp.27-35.
[10] R. Fielding et al, RFC2616 Hypertext Transfer Protocol -- HTTP/1.1, http://www.rfc.net/rfc2616.html, The Internet Society, 1999 .
[11] T. Ylonen, SSH - Secure login connections over the internet, Proceedings of the 6th Security Symposium, p 37, 1996.
[12] S. F. Altschul et al., Basic local alignment search tool, J. Mol. Biol. 215:403-10, 1990.
[13] G. Barish and K. Obraczka, World Wide Web Caching: Trends and Techniques, IEEE Communications Magazine Internet Technology Series, May 2000.
[14] Minimum intrusion Grid - The Simple Model, Henrik H Karlsen and Brian Vinter, in proc. of ETNGRID 2005 (to appear)
[15] Transparent Remote File Access in the Minimum Intrusion Grid, Rasmus Andersen and Brian Vinter, in proc. of ETNGRID 2005 (to appear)
[16] The Community Scheduler Framework, http://csf.metascheduler.org, 2005.
[17] The Nordic Data Grid Facility, NDGF, www.ndgf.org, 2003.
[18] Data Clinic, http://www.dataclinic.co.uk/data-backup.htm