Minimum intrusion Grid - The Simple Model

Henrik Hoey Karlsen
University of Southern Denmark
karlsen@imada.sdu.dk

Brian Vinter
University of Southern Denmark
vinter@imada.sdu.dk

Abstract: *This paper describes the key ideas in the new Grid model: MiG (Minimum intrusion Grid). "The Simple Model" is a major milestone in developing the full MiG concept and when completely implemented "The Simple Model" includes a full featured Grid solution. Besides describing the general MiG idea, the design of "The Simple Model" is specified as well as the main considerations behind the implementation.*

1. Introduction

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, is 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.

The driving idea behind the Minimum intrusion Grid, MiG, project is to develop a Grid 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.

1.1. Philosophy behind MiG

"The Minimum intrusion Grid", this really is the philosophy. We want to develop a Grid middleware that imposes as few requirements as possible. The working idea is to ensure that a user need only a signed X.509 certificate, trusted by Grid, and a web-browser capable of secure HTTP, HPPTS[6]. A resource on the other hand must also hold a trusted X.509 certificate and in addition create a user – the Grid user – who can use secure shell, ssh, to enter the resource[1] 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

Having minimum intrusion on both clients and resources can not be obtained using the classic client-server model that many current grid implementations make use of. A magic blackbox is introduced, making the abstract MiG model look like this:

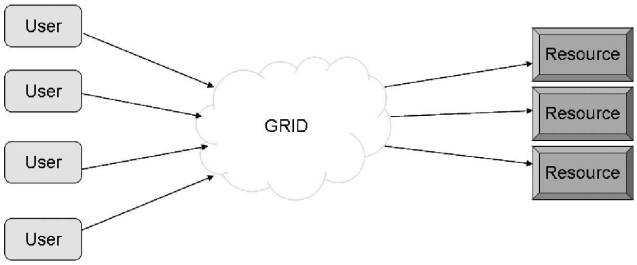

Figure 1. The abstract MiG model

Instead of clients and resources communicating directly they contact the centralized Grid black-box. This model allow us to remain full control of the Grid, thus upgrades and troubleshooting can be performed locally on the Grid instead of relying on collaboration from a large number of system-administrators. In addition, moving as much functionality as possible from the clients and resources to the 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 join the Grid.

1 A model that does not require ssh access to the user-account is also being investigated.

2. The Simple Model

In the simple model the centralized MiG black-box is replaced with a single MiG server:

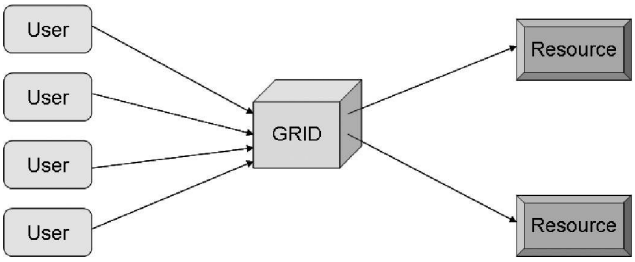

Figure 2. The Simple Model

In the simple model, clients and resources communicate with a single machine known as the central MiG server. It is always the clients and resources that initiates the communication. In the next three subsections the model is described as seen from the client, resource and the MiG servers point of view.

2.1. Client

One of the main design criterias is to make as few requirements as possible on the user. A web browser is available for almost every system and most users are familiar with how web browsers work. Therefore it has been decided that users should be able to communicate with the MiG system using their favourite browser. Some users might want to script their Grid activities like e.g. job submission, so MiG should also supply a small set of scripts that can be executed from a command line.

MiG need to validate the clients, so the system isn't open to all Internet users. Furthermore to avoid people abusing the system, e.g. trying to hack the MiG system or the resources, we need to control the pool of valid users. The normal solution to solve this issue it to introduce certificates, and also the way it is done in MiG.

Summarizing, this means that the implementation should enable users to access the MiG system with their normal web browser, or scripts provided by MiG, and a valid MiG certificate.

2.1.1. Resource Specification Language Like most Grid implementations, MiG also needs a resource specification from the user. To suit the MiG concept a new language has been created: mRSL, MiG Resource Specification Language. The keywords in the language have many similarities with Globus RSL[7] and NorduGrid/ARC xRSL[5], which is to be expected, since the purpose is the same. The new language has been designed with simplicity in mind, but provides all the necessary information.

2.2. Resource

MiG is designed to be as non-intrusive on the resources as possible. A resource in a Grid environment, however, will have some demands to the system and require some setup. It is important that we know the identity of the resources so we can distinguish resources from each other. In the future when accounting/banking is implemented it becomes even more important. This is the same problem as discussed in section 2.1, so the solution is the same: trusted certificates.

In addition to a valid certificate, resources need a small "MiG resource script". It is a small script that takes care of all the communication with the central MiG server. The script contacts the MiG server, asks for a new job, retrieves the job and the inputfiles, executes the job, uploads the outputfiles and nothing else. The "MiG resource script" can be submitted to the local queue system, and if no queue system is available it can be put in a wrapper script that loops forever and executes the script.

2.2.1. Configuration Each resource has its own private configuration. The configuration needs to be set before requesting the first job. Things like the available number of CPU's, amount of memory and diskspace is specified in the configuration file.

2.3. MiG server

One of MiG's main goals is minimum intrusion on clients and resources. As much code and many dependencies as possible have been moved to the central MiG server. This increases the requirements to the MiG server and the following list states the main tasks it must be able to solve:

- validate users using certificates
- provide web page where users can do common operations; submit jobs, send input files, monitor status of submitted jobs, retrieve outputfiles, list personal files
- receive mRSL based job descriptions from users
- validate resources using certificates
- provide a web page where resources can enter their configuration and view statistics
- send jobs to resources
- receive outputfiles from finished jobs from resources
- receive configuration files from resources
- parse mRSL files
- parse resource configuration files
- schedule jobs on resources

- notify users when their job is done

Looking at the list it is obvious that a web server is required to let users and resources have a private web page. Most up-to-date web servers support certificates - both "host certificates" and "client certificates". Installing a "host certificate" on the MiG server enables users and resources to verify it, and if the web server requires clients and resources to provide a "client certificate", the MiG server can check them too. A web server with the mentioned certificate support and support for file up- and download can do almost every item on the list. Parsing of mRSL files and resource configuration files, scheduling and notification of users is not directly supported by a web server, but the webserver can initiate external programs when needed.

2.3.1. Status The design of "The Simple Model" is considered done. Small changes might be introduced, but the main principle should stay as described in this section.

3. Implementation

This section describes the main considerations behind the implementation.

3.1. Overview

Certificates are an important element in the MiG system as discussed in section 2. The most used type of certificates is "x.509" [3] certificates and this type of certificate contains all the neseccary information needed by the MiG system. Because this certificate type is so widely used, it is constantly being considered if it is secure and it is supported on all major systems dealing with certificates. Furthermore x.509 certificates can easily be converted to .P12 format, which is the certificate standard used in webbrowsers. The x.509 certificate system can do everything needed in the MiG system and the mentioned advantages makes it the type of certificate that MiG uses on both the user, resource and the central MiG server.

Security is an important issue in all Grid systems. One of the basics is to encrypt network traffic to avoid eavesdropping on the communication line. HTTPS [6] is a widely used standard that provides secure communication using secure socket layer (SSL) [1]. It is supported in most webservers and webbrowsers making it a perfect solution to be the network protocol used in the MiG system. Many of the existing Grid systems have a lot of problems because their code is implemented in many different programming languages. It has been decided that all MiG code should be written in HTML or Python to avoid similar problems. Python is chosen because it has a very clear syntax, scripting support and many low-level considerations are hidden from the programmer, making developing in Python very fast compared with languages like C and C++.

The below figure shows the design of the MiG system:

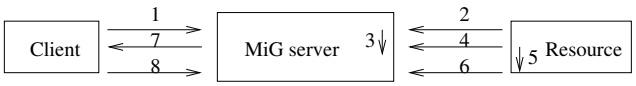

Figure 3. MiG implementation

1. A user submits inputfiles or a mRSL job description to the MiG server.
2. The resource requests a job.
3. The MiG server runs its scheduler and checks if a mRSL job description can be matched with the job request from the resource. If a match exists, then a job script is generated in the script language specified by the resource. If no suitable job exists in the job queue or the queue is empty, then a "empty job" is created containing 'sleep for *n* seconds' in the execute field to avoid the MiG server being spammed with requests.
4. The resource retrieves the job script.
5. The resource executes the job.
6. The resource uploads outputfiles to the MiG server.
7. The MiG server notifies the user by mail or a jabber[2] message, that the job has been done. This stage is optional for the user.
8. The user retrieves the files from his personal Grid directory if it is needed for local processing.

One of the central points in the MiG design is that clients and resources always initiate the communication with the MiG server, apart from the job completion announcement. The MiG server never contacts clients or resources, so clients and resources only needs to have outbound traffic allowed on the default HTTPS port in their firewall.

3.2. Client

Recalling section 2.1 there is only two elements to consider on the clients:

- Certificate
- Web Browser or scripts provided from MiG

It has already been decided in section 3.1 to use x.509 certificates. When a user joins MiG he receives a standard

2 Other messaging protocols might be implemented if requested

x.509 certificate (split in two files, public certificate and private key) and the certificate converted to .P12 format. P12 is the certificate format used in modern webbrowsers.

The user has two opportunities when wanting to communicate with the central MiG server. If the user wants to use a webbrowser the only requirement is that it supports host certificates and client certificates in .p12 format. Widely used browsers having these prerequisites includes Mozilla Firefox, Microsoft Internet Explorer and Netscape. On the web page the user can enter and submit mRSL job descriptions in a textarea, upload inputfiles, download outputfiles from finished jobs and view the status of all jobs submitted to MiG.

Alternatively the user can also choose to fetch and use the scripts provided from MiG. At the moment these scripts are implemented:

- MiGsubmit: Submits an input file to MiG or a mRSL file for execution
- MiGlist: List the contents of the users private directory
- MiGget Retrieve files from the users private directory
- MiGremove: Remove files from users private directory
- MiGstatus: Get status of a submitted job

The implementation of the scripts are very simple because of the MiG design. In practise they are all just wrappers around the "curl" [2] command line utility. "curl" is used to transfer files with URL syntax and supports HTTPS. It is available for nearly all systems and can be installed without root privilegies on Unix systems.

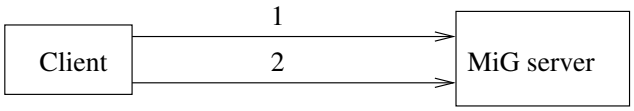

Figure 4. 1: User submits inputfiles and jobs to the MiG server. 2: User requests outputfiles and various information like job status from the MiG server.

Because the scripts are that simple it is highly unlikely it will be necessary to make changes to the scripts. This is a major advantage compared with most of the existing Grid middleware, where users often must upgrade their large software installations due to new features or security updates.

To show the simplicity of the client scripts, the pseudo code of MiGsubmit is shown here:

```
if number_of_arguments == 1
  filename = argument_#1
  execute curl --upload-file filename
    https://MiG_central_server_url/filename
  if execute_return_code == 0
     return ''file was submitted''
  else
     return ''file was NOT submitted''
```

Looking at the pseudo code it is easy to see that MiGsubmit is just a wrapper around the curl command. The same goes for all the other MiG scrips. The scripts are currently only implemented in the "sh" shell script language, but it is very simple to port it to windows .bat files and other script languages.

3.2.1. mRSL job specification language As stated earlier in this paper, users specify their job in the job description language mRSL. The language is not fully specified yet, but all the main functionality is available. To introduce the language a short example is shown:

```
::JOBNAME::
examplejob
::EXECUTE::
povray torus1.pov
::NOTIFY::
my@email.com
jabber: myaccount@jabber.com
::ENVIRONMENT::
MYENV=/home/user/
::EXECUTABLES::
executescript.sh
::INPUTFILES::
torus1.pov
::OUTPUTFILES::
torus1.png
::MEMORY::
128
::DISK::
10
::CPUCOUNT::
1
::ARCHITECTURE::
i386
::CPUTIME::
20
::RUNTIMEENVIRONMENT::
POVRAY3.6
```

The job description gives the job the friendly name "examplejob", and states that it should be run on a resource with at least 128 mb memory, 10 mb of free disk space, a single cpu with i386 architecture and the resource should offer its resources for at least 20 seconds. The resource should also have the runtimeenvironment POVRAY3.6 available. When the job arrives to the resource the execution is not

started before the files torus1.pov and executescript.sh are downloaded from the central MiG server (Another MiG project currently being developed is providing on-demand, remote transparent file access, making it possible to wait downloading the inputfiles until they are actually needed and only download the parts needed from the inputfile). When the files have been downloaded the resource turns on the executable mode of executescript.sh, and sets the environment MYENV=/home/user/. Then the resource executes the command 'hostname' and then 'povray torus1.pov'. When the resource has finished executing the commands the file torus1.png is uploaded to the central MiG server. Three special files are always uploaded, namely the files called .stderr, .stdout and .status. The .stderr file contains the output written to stdout during execution of the commands, stderr the output written to stderr and the status file lists the commands executed and the return code. When the MiG server receives a .status file it knows that the job is done and the MiG server sends a mail to my@email.com and a jabber message to myaccount@jabber.com.

These keywords have not been implemented yet, but are considered to be a part of the mRSL language:

- NodeMiGUnits - The power of the nodes on the resource measured in MiGUnits[3]
- OS - The operating system required

3.3. Resource

The figure shows the simple model from the resources point of view:

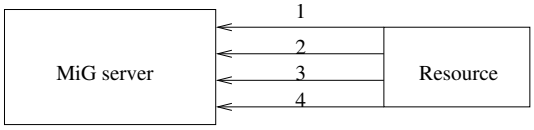

Figure 5. 1: The resource requests a job script 2: The resource retrieves the job script 3: The resource retrives the inputfiles specified in the job script 4: The resource uploads outputfiles to the MiG server

The core of the functionality on the resources is included in the job script generated on the MiG server. The job script is described in more detail in section 3.4.2. There are two requirements to a resource: It must have curl installed since it is being used by the job script to transfer files to and from the MiG server and it must have the very simple "resource script". The job script performs all operations needed to carry out a single job. The resource script takes care of requesting a job script from the MiG server, waits until it has been generated, downloads and executes it. Running the resource script once is the same as carrying out a single job and therefore a resource that wants to perform several jobs needs another script around the resource script. If the resource has a queue system e.g. PBS like many large clusters, it can be demonstrated like this:

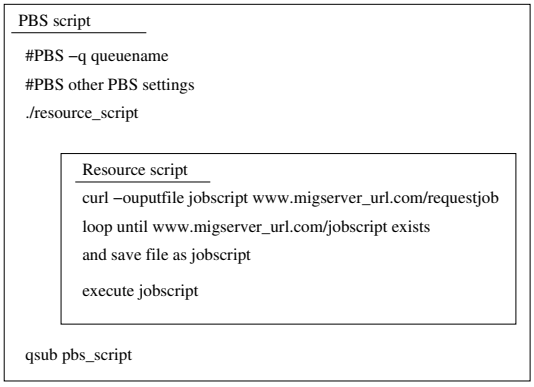

Figure 6. PBS script and resource script

The PBS script sets the PBS attributes and then executes the resource script. When the resource script is done, the PBS script is submitted to the queue again, making it an infinite loop. If no queue system is installed the pbs script is replaced with a script looping the resource script forever.

3.3.1. Configuration Every resource has its own configuration file. Here is an example of how it may look:

```
::SCRIPTLANGUAGE::
sh
::ARCHITECTURE::
i386
::MEMORY::
512
::DISK::
80000
::CPUCOUNT::
1
::RUNTIMEENVIRONMENT::
name: POVRAY3.6
name: LOCALDISK
```

The resource wants the job script in the scriptlanguage sh. At the moment sh and Python can be requested. Architecture, amount of memory, disk and cpu's are specified and the resource says it can execute jobs that needs runtimeenvironments POVRAY3.6 and LOCALDISK.

3 A number similar to "flops" but calculated in a different way

MiG provides a tool[4] to help resource administrators fill out their configuration file. The tool automatically identifies the available amount of memory, disk etc. and enters the information in the configuration file. Resource administrators are not required to use the tool, it is merely an option. When the configuration template has been filled out it can be send to the central MiG server using the MiG resource script. Resource administrators can also visit their private web page using their favourite browser where they can enter their current configuration as well as see various statistics, number of jobs processed etc.

3.4. MiG server

The central part of the MiG server is the webserver. According to [4] the worlds most widely used webserver is the Apache httpd server. The web server is fast, modular and has all the required functionality the MiG webserver needs including support for host and client x.509 certificates. Because the webserver is so popular and used in such a large number, security patches is immediately available whenever a security hole is found. This makes the Apache web server suitable for the MiG central server. With time a dedicated MiG server will be developed to replace the generic webserver.

3.4.1. Scheduler Users submits jobs to the MiG server and resources requests jobs to execute from the MiG server. The users specify the details about the systems their job should be ran on in the mRSL file and resources specify what they offer in their configuration file. When a job is received on the MiG server, it is put into a jobqueue, and when a resource requests a job the MiG server need to schedule a job for the resource. At the moment the scheduling algorithm implemented is very simple: first-fit scheduling.[5]

3.4.2. Job script generation When the job has been scheduled to be run on a specific resource the job script generator is started. The script is created in the script language specified in the ::RUNTIMEENVIRONMENT:: attribute in the specific resources configuration. The script is based on the scheduled mRSL file and performs all the requested actions in the job description. When the job script is generated it contains the following actions:

```
Create job directory
Enter job directory
Get Inputfiles
Get Executables
Mark executables as executable
Set environments
Set runtimeenvironments
Execute the specified commands
Submit outputfiles
Send special files
Leave job directory
Clean up
```

Locating the job script generator on the MiG server helps keeping down the intrusion on the resource - without exception are as much code as possible placed on the MiG server.

4. Performance

The implementation is not yet fully completed, thus only the core parts of "The Simple Model" can be benchmarked. One of the most important aspects of a Grid system is the throughput it can handle. This indicates if the solution is scalable and how many users the system can handle with acceptable performance.

The implementation of The Simple Model is compared with a NorduGrid/ARC installation; client version 0.4.4 and server version 0.5.14. The NorduGrid/ARC software is a very popular middleware based on the Globus toolkit and runs in several production environments, with more than 5000 CPU's connected in total.

The MiG setup:
Clients, Intel Pentium 4 2.4ghz
MiG server, Intel Celeron 2.6ghz
Resources, Intel Pentium 4 2.4ghz
mRSL file used (helloGrid.mRSL):

```
::EXECUTE::
echo ''hello Grid''
```

Command used: MiGsubmit helloGrid.mRSL

The NorduGrid/ARC setup:
Client, Intel Penium M 1.5ghz
Resource, Intel Pentium 4 2.66ghz
Command used: ngtest -c resource_url 1

The submit times are plotted in figure 7.

The MiG system is tested with one and five clients submitting jobs to respectivly no resources or five resources requesting jobs to execute. In the NorduGrid/ARC test there are a single client submutting jobs and one resource executing the jobs. It has not been possible to make a NorduGrid/ARC setup with more than one client and one resource. The fastest submit times are MiG with five users each submitting a fifth of the specified number of jobs and no resources asking for jobs. The next best is MiG with one user and no resources followed by MiG with one user and one resource asking for jobs. This comes as no surprise since the resources will consume some of the servers resources. It takes a bit longer per job when five resources is

4 Not fully implemented yet

5 A separate project to develop a Grid-level scheduler has been started and within the nearest future a new, better scheduling algorithm will be implemented.

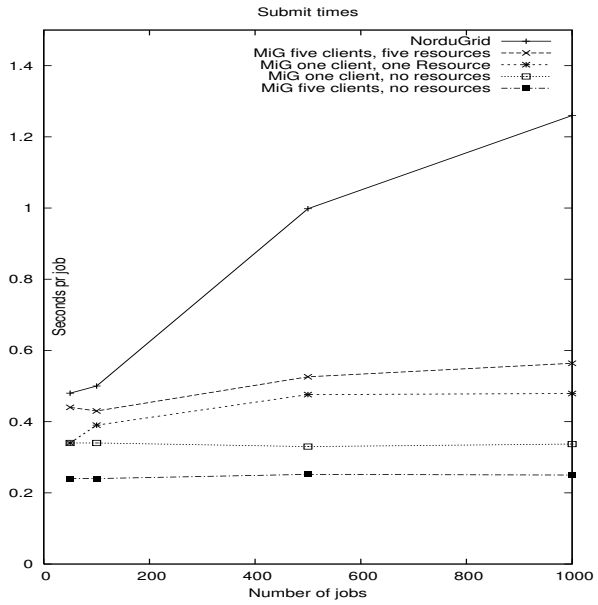

Figure 7. Submit times

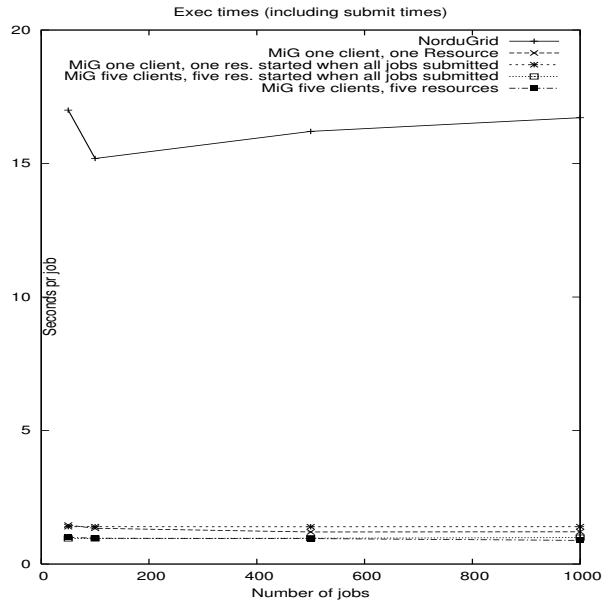

Figure 8. Exec times

interrupting the MiG server with job requests and the slowest result is NorduGrid/ARC with one client and one resource. When comparing MiG and NorduGrid/ARC submit times the comparable MiG graph is the one labelled "MiG one user, one resources". With one client and one resource the MiG system has minimum 10 % higher throughput.

The "exec" time is the time from the first job is submitted until the resource has executed the last job and finished uploading the last .status file in MiG and stdout file in NorduGrid/ARC. The MiG system has been tested where the resources are connected from the beginning of the test and when the resources are started when the last job has been submitted. In the NorduGrid/ARC test the resource starts to execute the job when it is received, which means it should be compared to the MiG graph labelled "MiG one client, one resource".

The big difference in the "exec times" (figure 8) makes it impossible to find a good scale for the graph. The MiG system is an order of magnitude faster than NorduGrid/ARC when comparing the complete executing time. To be able to see the various MiG numbers a new graph is needed (figure 9)

The time taken with five clients and five resources connected from the beginning is lower than a single client with a single resource connected. In both cases where the resources is started from the beginning, the times are a bit faster than when the resources are first started when the last job has been submitted, but the difference is not as big as it might have been expected.

The benchmark of the MiG systems throughput shows that it can easily compete with a Grid middleware that runs in several production environments. The throughput of the MiG system is stable, no matter if just a single user is served or five users and five resources simultaneously. Actually the NorduGrid/ARC model will have worse numbers when the number of connected resources increases, since it doesn't have the central knowledge as MiG, and therefore a client needs to contact all resources before it can submit the job. The more resources connected the longer it takes to submit a job in a Grid like NorduGrid/ARC, but in MiG the throughput is approximately the same. It is important to stress that NorduGrid/ARC is a widely used and accepted middleware, but this performance test shows that the MiG design can compete with the globus-like client-server design.

5. Future work

5.1. Implementation

The implementation of "the simple model" will continue until it has reached a level where it is ready for a production environment.

5.2. Unresolved issues

There are a few unresolved issues, which have not been solved yet. At the moment the biggest is how to keep the private key from the resource's certificate secure from malicious users. Inputfiles and outputfiles must be send using HTTPS with certificates, but this means that a user can specify the resource certificate private key as an outputfile! To avoid this one solution could be to require from users that

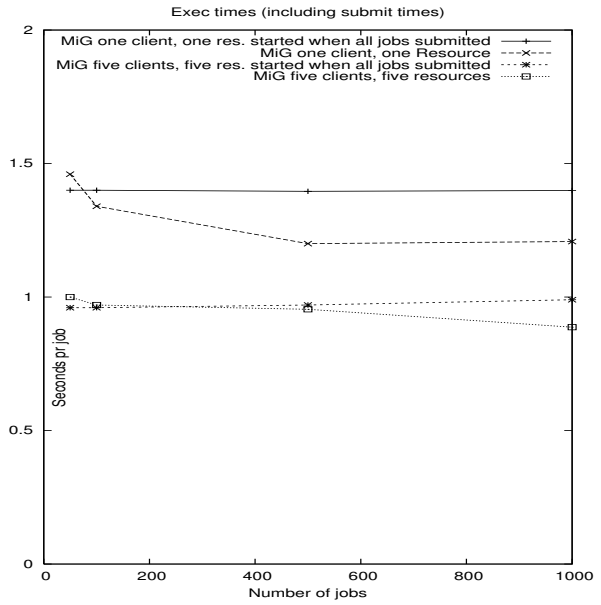

Figure 9. Exec times without NorduGrid/ARC

the part of the script that communicates with the MiG server is another user than the one executing the script. This is the solution in most current grid systems, where root executes the script and the script changes the user under the execution of the user-specified executables. Another solution, using the resources ssh hostkey as identification to the MiG server is also possible. Instead of requiring two users on the resource this will make the requirement that the resource has a ssh server running.

5.3. The Full MiG model

It is obvious that the simple model has a number of shortcomings. The MiG server is a single point-of-failure. If the MiG server is down due to eg. a crash or maintainance there is no working Grid. With a single MiG server the load on its resources might be too high and this might result in different forms of lag and could cause the service to be unavailable. Furhermore it will be very vulnerable to a denial of service attack (DoS) since attackers can use all their bandwidth in attacking this single point. The solution is just as obvious as the problem: to introduce more MiG servers. It is important that it is transparent to the users and resources that there are multiple MiG servers. From the users perspective it must still look like a single Grid. This solution with multiple Grid servers is known as "the full model" and will be implemented when the simple model is done. The full model solves all the shortcomings of the simple model, has more features than the traditional grid systems and the advantages of minimum intrusion on users and resources.

6. Conclusions

The purpose of this paper is to describe "the simple model", a major milestone in developing the entire MiG concept. When fully implemented, "the simple model" is a complete, working, full-featured Grid system with many advantages compared with the traditional client-server based Grid implementations.

The design of "the simple model" is described as seen from the user, resource and Grid's point-of-view. Many implementation details are discussed giving the reader a good impression of the requirements to users and resources and how the system will work when the ongoing implementation has been done and the Grid system is put into production. A benchmark is presented, where the throughput of the system with various levels of load is compared with a NorduGrid/ARC setup. Even though the implementation of "the simple model" is far from done and some improvements to the system are still missing, "the simple model" can easily compete with the Nordugrid in the time taken to submit jobs, and in the time taken for a job to go trough the entire system "the simple model" is an order of magnitude faster than the NorduGrid/ARC software.

No major problems with the MiG concept have been found during the design and implementation and it still looks like it represent a significant improvement in the way Grid middleware is designed.

References

[1] a. e. Alan O. Freier. "the ssl protocol". "http://wp.netscape.com/eng/ssl3/draft302.txt".

[2] cURL. "curl - tool for transferring files with url syntax". "http://www.curl.haxx.se".

[3] International Telecommunication Union. "recommendation x.509". "http://www.itu.int/rec/recommendation.asp ?type=folders&lang=e&parent=T-REC-X.509".

[4] Netcraft. "december 2004 web server survey". "http://news.netcraft.com/archives/web_server_survey.html".

[5] NorduGrid. "http://www.nordugrid.org".

[6] a. e. R. Fielding. Rfc2616 hypertext transfer protocol – http/1.1. "http://www.rfc.net/rfc2616.html".

[7] The Globus Alliance. "http://www.globus.org".