

# Managing MPICH-G2 Jobs with WebCom-G

Padraig J. O'Dowd, Adarsh Patil and John P. Morrison  
Computer Science Dept.,  
University College Cork,  
Ireland  
{p.odowd, adarsh, j.morrison}@cs.ucc.ie

**Abstract**—This paper discusses the use of WebCom-G to handle the management & scheduling of MPICH-G2 (MPI) jobs. Users can submit their MPI applications to a WebCom-G portal via a web interface. WebCom-G will then select the machines to execute the application on, depending on the machines available to it and the number of machines requested by the user. WebCom-G automatically & dynamically constructs a RSL script with the selected machines and schedules the job for execution on these machines. Once the MPI application has finished executing, results are stored on the portal server, where the user can collect them. A main advantage of this system is fault survival, if any of the machines fail during the execution of a job, WebCom-G can automatically handle such failures. Following a machine failure, WebCom-G can create a new RSL script with the failed machines removed, incorporate new machines (if they are available) to replace the failed ones and re-launch the job without any intervention from the user. The probability of failures in a Grid environment is high, so fault survival becomes an important issue.

*Keywords:* WebCom-G, Globus, MPICH-G2, MPI, Grid Portals, Scheduling and Fault Survival.

## I. INTRODUCTION

Grid Computing has received much attention recently as it offers users the ability to harness the processing power of a large number of resources. The Globus Toolkit [4] has become the de-facto software for building grid computing environments and MPICH-G2 [7] uses Globus to provide a grid-enabled implementation of the MPI v1.1 standard. Being able to run unmodified legacy MPI code in this manner, strengthens the usability of grid computing for end users. However, there are some disadvantages/limitations to running MPICH-G2 jobs with RSL scripts (e.g., little or no support for job recovery after failure) and these will be discussed in detail in Section III. This paper proposes using WebCom-G to address some of these issues, such as automating the deployment, execution & fault survival of MPICH-G2 jobs and extends work proposed in [9]. Other work [10] has investigated the use of WebCom-G for handling issues like fault survival with MPICH and running MPI jobs in a Beowulf cluster environment.

The remainder of this paper is organised as follows: Globus and its Execution Platform is discussed in Section II. MPICH-G2 and how it uses Globus is described in Section III. The WebCom-G Grid Operating System is presented in Section IV. In Section V, the control of MPICH-G2 by WebCom-G is discussed and how WebCom-G can ensure the fault survival of MPICH-G2 jobs. In Section VI, some sample executions

and results, achieved from testing the system are presented. Finally; Section VII presents conclusions and future work.

## II. GLOBUS AND ITS EXECUTION PLATFORM

Globus [4] provides the basic software infrastructure to build and maintain Grids. As dynamic networked resources are widely spread across the world, information services play a vital role in providing grid software infrastructures, discovering and monitoring resources for planning, developing and adopting applications. The onus is on the Information services to support resource & service discovery and subsequently use these resources and invoke services. Thus the Information services is a crucial part of any Grid.

An organisation running Globus hosts their resources in the Grid Information Service (GIS), running on a Gatekeeper machine. The information provided by the GIS may vary over time in an organisation. The information provider for a computational resource might provide static information (such as the number of nodes, amount of memory, operating system version number) and dynamic information such as the resources uncovered by the GIS, machine loads, storage and network information. Machines running Globus may use a simple scheduler or more advanced schedulers provided by Condor, LSF, PBS and Sun Grid Engine.

Typically users submit jobs to Globus (2.4) by means of a Resource Specification Language (RSL) script executing on the Gatekeeper, provided they have been successfully authenticated by the Grid Security Infrastructure(GSI). The RSL script specifies the application to run and the physical node(s) that the application should be executed on and any other required information. The Gatekeeper contacts a job manager service which in turn decides where the application is to be executed. For distributed services, the job manager negotiates with the Dynamically Updated Request Online Co-allocator (DUROC) to find the location of the requested service. DUROC makes the decision of where the application is to be executed by communicating with each machines' lower level Grid Resource Allocation Manager (GRAM). This information is communicated back to the job manager and it schedules the execution of the application according to its own policies. If no job manager is specified, then the default service is used. This is usually the "fork" command, which returns immediately. A typical grid configuration is shown in Fig. 1.

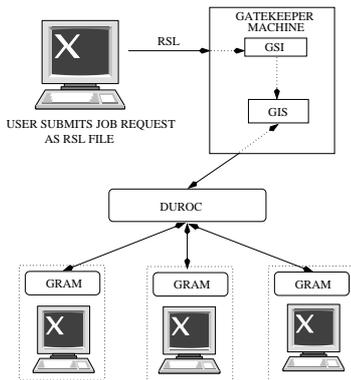


Fig. 1. Globus Execution model. A user generates an RSL script and submits it to the Gatekeeper. The Gatekeeper, in conjunction with DUROC and GRAM facilitate the distributed execution of requested services on the underlying nodes.

There are some disadvantages to using RSL scripts. Most notably, in a distributed execution if any node fails, the whole job fails and will have to be re-submitted by the user at a later time. There is no diagnostic information available to determine the cause of failure. Only resources known at execution time may be employed. There is no mechanism to facilitate job-resource dependencies. The resource must be available before the job is run, otherwise it fails. There is no in-built checkpointing support, although some can be programmatically included. This may not be feasible due to the particular grid configuration used. Also, RSL is only suited to tightly coupled nodes, with permanent availability. If any of the required nodes are off-line, the job will fail.

### III. MPICH-G2

MPICH-G2 [7] is a grid-enabled implementation of the MPI v1.1 standard. It uses services from the Globus Toolkit to handle authentication, authorisation, executable staging, process creation, process monitoring, process control, communication, redirecting of standard input (& output) and remote file access. As a result a user can run MPI programs across multiple computers at different sites using the same commands that would be used on a parallel computer or cluster. MPICH-G2 allows users to couple multiple machines, potentially of different architectures, to run MPI applications. It automatically converts data in messages sent between machines of different architectures and supports multi-protocol communication by automatically selecting TCP for inter-machine messaging and (where available) vendor-supplied MPI for intra-machine messaging. According to [7], performance studies have shown that overheads relative to native implementations of basic communication functions are negligible.

As shown in Fig. 2, MPICH-G2 uses a range of Globus Toolkit services to address the various complex situations that arise in heterogeneous Grid environments. MPICH-G2 was created by creating a 'globus2' device for MPICH [5]. MPICH supports portability through its layered architecture. At the top is the MPI layer as defined by the MPI standards. Directly underneath this layer is the MPICH layer, which implements

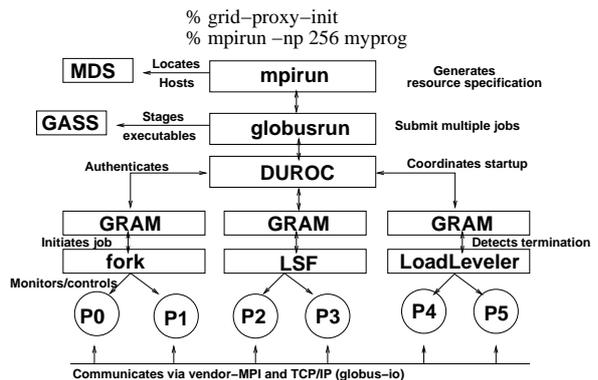


Fig. 2. Overview of the startup of MPICH-G2 and the use of various Globus Toolkit components to hide and manage heterogeneity. (This diagram was taken from [7])

the MPI interface. Most of the code in an MPI implementation is independent of the underlying network communication system. This code, which includes error checking and various manipulations of the opaque objects, is implemented at the MPICH layer. All other functionality is passed to lower layers by means of the Abstract Device Interface (ADI). The ADI is a simpler interface than MPI proper and focuses on moving data between the MPI layer and the network subsystem. Those wishing to port MPI to a particular platform need only define the routines in the ADI in order to obtain a full implementation. This is how MPICH-G2 works, by creating a special ADI device that uses the Globus Toolkit.

### IV. WEBCOM-G

The WebCom-G Grid Operating System is a multi-layer platform for executing distributed applications on a large number of dispersed resources. Applications are separated from the underlying computing infrastructure and are specified as Condensed Graphs [8]. Condensed Graphs are used to specify tasks and the sequencing constraints associated with them. WebCom-G supports fault tolerance/survival, load balancing, scheduling and security at different levels within its architecture. WebCom-G seeks to provide Grid access to non-specialist users and from the perspectives of application developers and end users, to hide the underlying Grid. An in-depth discussion of WebCom-G is beyond the scope of this paper, for this readers are referred to [6].

### V. MANAGING MPICH-G2 JOBS WITH WEBCOM-G

Like other middlewares, WebCom-G provides brokerage for underlying resources and provides a single system image to the end user. Interfaces to these middlewares are made in several ways, e.g. Web Services and Grid/Web Portals (where users visits a web-page, uploads applications, these are run across a heterogeneous set of machines and the results are returned).

Presently, there are many Grid Portal development kits that are aimed at developing Grid Portals. Some of these portals are application specific and some of them are Globus based Portals. Unlike some other Grid middlewares, WebCom-G

is modularised, it uses a plug-in mechanism to extend its functionality and indeed plug-ins have been developed to incorporate some other middlewares such as Globus, EJB, COM, DCOM and Corba. Currently plug-ins are being developed for NetSolve [3] and DIET [1]. Applications specified as Condensed Graphs, whose nodes represent services of these middlewares are scheduled in a fault tolerant manner, exploiting in-built scheduling and security mechanisms by WebCom-G.

#### A. Automating the Deployment, Execution & Fault Survival of MPICH-G2 jobs

Authorised users can visit a WebCom-G Portal (e.g., <http://portal.webcom-g.org>) and submit their applications, which are written in the form of Condensed Graphs. WebCom-G then schedules the execution of these (applications represented as) graphs across the available resources. As well as the end results, various statistics about each job and the current state of the underlying resources (see Fig 3) are also maintained by the Portal.

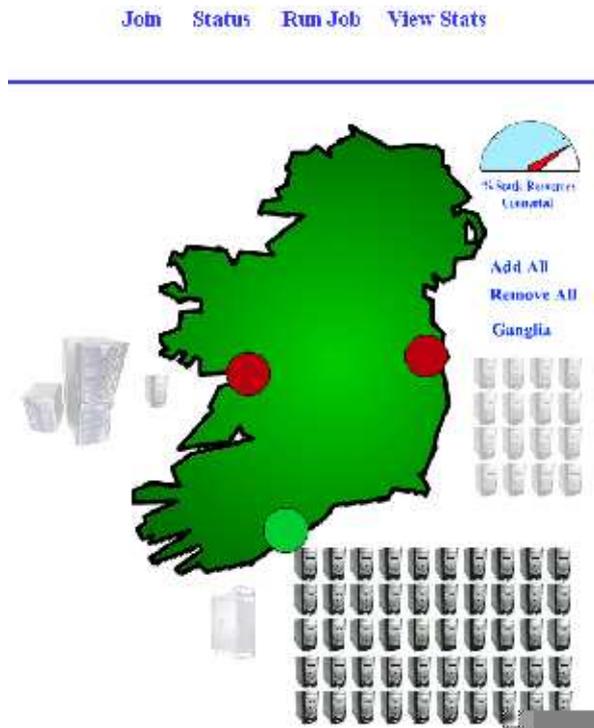


Fig. 3. Graphical view of the resources connected to the WebCom-G Portal

The above notion was extended for allowing portal users to execute MPI (MPICH-G2) jobs on the WebCom-G Portal (and the WebCom-G Grid). Users can visit the WebCom-G Portal, upload their MPI code and specify the number of machines they require. WebCom-G firsts decides the machines on which the MPI application will execute. The MPI code is then compiled and executed on these machines. If any errors occur during the compilation, they are returned to the user and the process aborted. Finally the application is scheduled for

execution on the underlying resources and the following tasks are performed:

- A Resource Specification Language (RSL) script is built on the fly, depending on machine availability, path & package availability.
- The application is run using the RSL file.
- In the case of failure, the application is rescheduled with a new RSL file, dynamically created by WebCom-G from a new interrogation of the underlying resources. WebCom-G then re-launches the job with the new RSL file. No user intervention is required at this point.
- Finally results are sent to the Portal, for collection by the user.

#### Overview of the MPICH-G2 Condensed Graph Application

This section presents an overview of the MPICH-G2 Condensed Graph application that was developed, in order to use WebCom-G to manage MPICH-G2 jobs (i.e. handling issues like fault survival etc). The WebCom-G IDE and WebCom-G Interrogators are used extensively in this application, but an in-depth discussion on these is beyond the scope of this paper. The following presents a summary on them:

- **Interrogation:** When WebCom-G is running on a machine, it can run it's own interrogators to see what services the machine is running and then register these services with an Interrogator Database. The Portal or any WebCom-G machine in general can use this Interrogator Database for scheduling issues e.g., to see what machines are running a certain service. A top-level WebCom-G machine (e.g., Portal Server) can request all it's client WebCom-G machines (and their clients, if they have any) to run their interrogators by simply executing a particular Condensed Graph application. This Condensed Graph is recursive in that when it executes on a machine it invokes the interrogator on that machine and passes an instance of itself onto all the clients of the machine for execution. So every client WebCom-G machine connected to the tree (that has the top level WebCom-G machine as root) register all their available services with the Interrogator Database.
- **WebCom-G IDE:** The WebCom-G Integrated Development Environment provides a graphical method for creating Condensed Graphs. From within the IDE a user can create, load, save and execute Condensed Graph applications. A palette of nodes is supplied; these can be dragged onto the canvas, and linked together to form the graph. Fig. 4 shows the MPICH-G2 application developed with the WebCom-G IDE. Also a WebCom-G IDE can query the Interrogator Database to see what services are available and then display these services on its palette. Users can then create Condensed Graph applications graphically in the IDE that can incorporate these services.

Fig. 4 shows the MPICH-G2 Condensed Graph application that was created with the IDE. For a detailed discussion of how the Condensed Model of Computing operates, readers

are referred to [8]. The following presumptions were made about the graph based on the Globus test-bed that was used in its development. All machines taking part used a shared file-system, and before the graph is executed the source will have been compiled (and will have to have compiled without errors, otherwise the process is aborted and the user is notified.) Though the graph can easily be changed to account for machines that don't use a shared file system by adding some nodes to the graph that transfer the MPI source code to each client machine and compiles it.

The graph takes five inputs:

- Number of machines required.
- MPI source code.
- Directory to run the MPICH-G2 job from.
- Arguments to pass to the MPICH-G2 job.
- Timeout: Users (if they want) are allowed to specify a timeout for their MPICH-G2 job, if the execution time of a job exceeds the timeout, the job is thought to have failed and is re scheduled.

Overview of the main nodes of the graph (see Fig. 4) and how it executes:

- *GetClientListOp* - This node takes two parameters, the name of a service, which in the case of this graph is mpichg2 and the number of machines required. The

node when executed outputs a list of machine names (no greater then the number of required machines). This node uses the current list of available client machines connected to the (WebCom-G) Portal and the Interrogator Database to see which of these clients can execute MPICH-G2 jobs. It then creates a list of machines that can be used in the execution of the MPICH-G2 job. In a properties file, users can set rules that effect how machines in the returned list are selected. For example, machines can be selected based on their load state - if one hundred machines are available in a WebCom-G Grid (that can execute MPICH-G2 Jobs), but only eighty are required to execute a particular job, then the eighty most lightly loaded machines of the one hundred are returned.

- *RSLGeneratorOp* - This node generates an RSL file which can be used to execute the MPICH-G2 job. It takes the list of machine names generated by the GetClientListOp node, the name of the executable. the directory to run the MPICH-G2 job from and the arguments to pass to the job. From these, this node generates the RSL Script.
- *Service3CG* - This node executes the MPICH-G2 job with the generated RSL Script and the timeout if specified by the user. This node then monitors the execution of the job

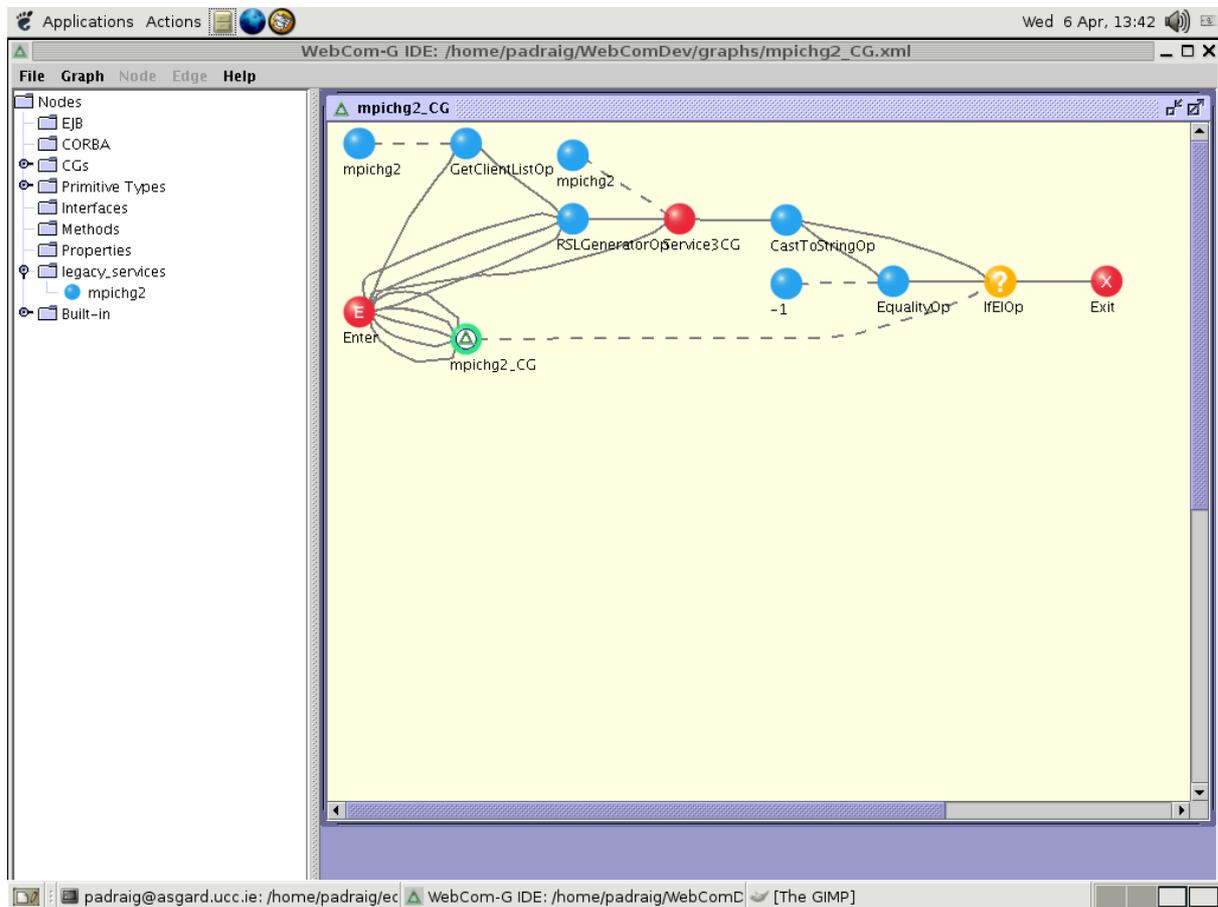


Fig. 4. The MPICH-G2 Condensed Graph Application developed in the IDE.

and in the event of a job failure/crashing, it detects the job failure and reports it. This node can detect failures of MPICH-G2 jobs in the following ways:

- *mpirun returns and data has been written to the "stderr"*: There are a few bugs in the current release of MPICH-G2, (see [2] for details) that affect the detection of failures from the mpirun command. One of these bugs is that "the exit code passed to MPI\_Abort does not get propagated back to mpirun". Another is that sometimes "when calling MPI\_Abort, stdout/stderr are not always flushed unless the user explicitly flushes (fflush) both prior to calling MPI\_Abort, and even then, the data is sent to stdout/stderr of the other processes". These bugs are due to be fixed in future releases of MPICH-G2. So as the Service3CG can not get the exit code passed back by MPI\_Abort to mpirun, it just monitors the "stderr", if any data is written to "stderr" the node considers the job to have failed. When users are submitting MPI jobs to the WebCom-G Portal, they are expected to be aware of these issues. When future releases of MPICH-G2 are released, these problems can be removed.
- *A WebCom-G Client (that is in the RSL script) fails*: Users (if they want) can specify that a job is considered to have failed if a WebCom-G machine which was included in the RSL script fails during the execution of the MPICH-G2 job. So when a WebCom-G client fails, the Service3CG forces the job to terminate and reports the job as failed. One potential problem with this type of fault detection is that, it could be possible that the WebCom-G Client that failed might have finished actively taking part in the job and so even despite its failure, there is no reason to reschedule the job.
- *Timeout*: Users (if they want) are allowed to specify a timeout for their MPICH-G2 job, if the execution time of a job exceeds the timeout, the job is thought to have failed and is rescheduled. This timeout is only supplied as an option and is not ideally suited to a Grid environment, as resources in the grid are heterogeneous in nature. So executing an MPICH-G2 job on different machines (of different performance levels), obviously can cause the execution time of the job to vary greatly between runs. Its presumed, if a user specifies a timeout that they are thinking of a worst case scenario.
- *IfEIOp* - When this node fires, if the MPICH-G2 job failed it will evaluate to true, in which case the *MPICHG2\_CG* node fires, this node is really just a recursive call to the graph itself and causes it to execute again. (When the graph is re-executed any machines that have failed with not be in the list generated by the *GetClientListOp* node. Users can set in a properties files to have a re-integration of the underlying resources

again in case any new resources have been added since the previous execution, though any new resources that join the Portal, usually perform interrogation when they join.) Otherwise the job will have executed correctly and the graph exits and returns the results to the Portal for collection by the user.

## VI. SAMPLE EXECUTIONS AND RESULTS

A simple application was developed to demonstrate the system in action (using WebCom-G to manage MPICH-G2 jobs) - the application simply counts all the prime numbers up to a certain number (1 million) and returns the result. The number partition (1 to 1 million) is divided evenly among the available machines - obviously with this type of application, its execution time decreases as more machines are added. No complex algorithms were used just a simple brute force check, as the goal was to highlight WebCom-G's management of MPICH-G2 jobs not the efficiency of the MPI program.

Our test-bed consisted of six Debian machines that had Globus, MPICH-G2 and WebCom-G installed. So when the Portal was scheduling MPICH-G2 jobs it could use these six machines.

The following executions were performed and their results recorded:

- Simulation 1: Execute the application on five machines (no failures).
- Simulation 2: Execute the application on four machines (no failures).
- Simulation 3: Execute the application on five machines & during it's execution one controlled machine failure occurred, causing WebCom-G to reschedule the job
- Simulation 4: Execute the application on five machines & during it's execution two controlled machines failures occurred, causing WebCom-G to reschedule the job

*Note*: As there are only six machines in the test-bed, if a user wants five machines but two fail then obviously the application can only be re-run with four. This is because our test-bed had just six machines, it's presumed that in a true Grid environment, more machines would be available then required by the user and if one of their allocated machines fails, a new one can be obtained in its place.

The '*Total Execution Time*' of the application includes the time from when 'mpirun' was called to when it returns. The '*Individual Execution Time*' includes the time each machine spent executing its own prime count on the number partition that was assigned to it. In the case of a controlled machine failure this time indicates the time from when it started until it failed. (Time is shown in seconds.) As the number space (1 to 1 million) is divided into number segments, the segments with smaller numbers with execute much faster then the segments with larger numbers. For example in the following experiments Machine 1 gets smaller numbers then Machine 5, so Machine 1 checks its assigned number space a lot faster then Machine 5. Also its worth noting that in Simulation 3 and 4, Machine 1 finished executing before the failures, but when the application was re-launched Machine 1 re-did the work it had already

completed - this is because the current system guarantees only fault survival. If the job fails before fully finishing, it is completely re-launched.

*Simulation 1 :Execute the application on five machines (no failures)*

- Total execution time: 232
- Individual execution time for each machine:
  - Machine 1: 28.3
  - Machine 2: 78.2
  - Machine 3: 123.5
  - Machine 4: 171.2
  - Machine 5: 207.9

*Simulation 2 :Execute the application on four machines (no failures)*

- Total execution time: 281
- Individual execution time for each machine:
  - Machine 1: 43.4
  - Machine 2: 120.5
  - Machine 3: 188.6
  - Machine 4: 262.5

*Simulation 3 : Execute the application on five machines & during it's execution one controlled machine failure occurred, causing WebCom-G to reschedule the job*

- Total execution time: 355
- Individual execution time for each machine -run 1:
  - Machine 1: 28.4
  - Machine 2: 78.6
  - Machine 3: 100
  - Machine 4: 100
  - Machine 5: (failed at) 100
- Individual execution time for each machine -run 2:
  - Machine 1: 28.2
  - Machine 2: 78.5
  - Machine 3: 123.3
  - Machine 4: 171.2
  - Machine 5: (new machine) 207.3

*Simulation 4 : Execute the application on five machines & during it's execution two controlled machines failures occurred, causing WebCom-G to reschedule the job*

- Total execution time: 411
- Individual execution time for each machine -run 1:
  - Machine 1: 28.3
  - Machine 2: 78.1
  - Machine 3: 100
  - Machine 4: (failed at) 100
  - Machine 5: (failed at) 100
- Individual execution time for each machine -run 2:
  - Machine 1: 43.4
  - Machine 2: 118.3
  - Machine 3: 188.3
  - Machine 4: (new machine) 256.4

(With the two machines failures, only four machines left)

### Summary

Fig. 5 shows the total execution times of the different simulations. Due to the simplicity of the application these results are, as would be expected:

- Simulation 1 takes 232 seconds to execute with five machines.
- Simulation 2 takes 281 seconds to execute with four machines obviously taking longer to execute then Simulation 1.
- Simulation 3 takes 355 seconds to execute, because during its first execution at time 100, machine five fails, causing the whole job to fail and be rescheduled. During its second execution the application completes fully. So the total execution time for this simulation is roughly around: 100 (time in the first execution when the job failed) + 232 (the total execution time from Simulation 1) + 23 (the overhead of running the graph in WebCom-G and the start-up of an MPICH-G2 job - the time 100 refers only to the time the program spent executing the prime count code, it does not include the startup time for the MPICH-G2 job).
- Simulation 4 takes 411 seconds to execute, because during its first execution at time 100, two machines (four and five) fail, causing the whole job to fail and be rescheduled. During its second execution the application completes fully (but with only four machines, as there were only six machines in our test-bed and two had failed). So the total execution time for this simulation is roughly around: 100 (time in the first execution will the job failed) + 281 (the total execution time from Simulation 2) + 25 (the overhead of running the graph in WebCom-G and the start-up of an MPICH-G2 job - the time 100 refers only to the time the program spent executing the prime count code, it does not include the startup time for the MPICH-G2 job).

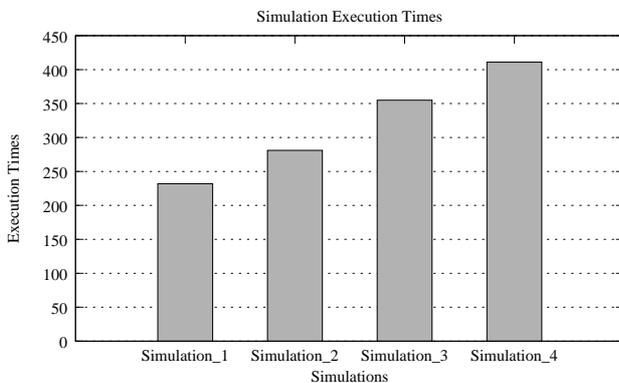


Fig. 5. Total execution times of the different simulations.

## VII. CONCLUSIONS

In this paper, the WebCom-G Grid Operating System was discussed and how it can be used to automate the deployment,

execution & fault survival of MPICH-G2 jobs. The main purpose of this research has been to focus on fault survival issues related to MPICH-G2 jobs and allow users to execute their jobs in grid environments, through the use of simple easy to use web interfaces. If a job submitted to the grid fails, the whole job will have to be re-submitted. However, the benefits of using WebCom-G for scheduling these jobs are immediately perceptible. If a job fails WebCom-G's fault survival will ensure that the job is automatically rescheduled. This may be to the same grid, or a different grid with the required resources. This dynamic scheduling of grid operations is possible by delaying the creation of the RSL script until just before it is required. Once the RSL script has been created, the physical configuration is determined.

#### REFERENCES

- [1] Distributed Interactive Engineering Toolbox. <http://graal.ens-lyon.fr/DIET/>.
- [2] [http://www.hpclab.niu.edu/mpi/g2\\_body.html#troubleshoot](http://www.hpclab.niu.edu/mpi/g2_body.html#troubleshoot).
- [3] S. Agrawal, J. Dongarra, K. Seymour, and S. Vadhiyar. NetSolve: Past, Present, and Future - a Look at a Grid Enabled Server.
- [4] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
- [5] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.
- [6] David A. Power, John P. Morrison, Brian Clayton and Adarsh Patil. Webcom-g: Grid enabled metacomputing. *The Journal of Neural, Parallel and Scientific Computation. Special Issue on Grid Computing.*, 2004(12)(2):419–438, April 2004.
- [7] N. T. Karohis. Mich-g2: A grid-enabled implementation of the message passing interface. <http://citeseer.ist.psu.edu/554400.html>.
- [8] John P. Morrison. *Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing*. PhD thesis, Technische Universiteit Eindhoven, 1996.
- [9] Adarsh Patil, Pdraig J. O'Dowd and John P. Morrison. *Automating the Deployment, Execution & Fault Survival of MPICH-G2 jobs with WebCom-G*. Cluster Computing and Grid (CCGRID), Cardiff, UK, May 9-12, 2005.
- [10] Brian Clayton, Therese Enright and John P. Morrison. *Running MPI Jobs with WebCom*, Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, USA, June 27-30, 2005.