High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?

David Abramson †, Jon Giddy‡ and Lew Kotler §

† School of Computer Science and
Software Engineering
Monash University,
CLAYTON, VIC 3168, Australia

§ Australian Radiation Protection and
Nuclear Safety Agency
Lower Plenty Road
YALLAMBIE VIC 3085, Australia

‡ CRC for Distributed Systems Technology
General Purpose South Building,
University of Queensland,
ST LUCIA, QLD, Australia

Abstract

This paper examines the role of parametric modeling as an application for the global computing grid, and explores some heuristics which make it possible to specify soft real time deadlines for larger computational experiments. We demonstrate the scheme with a case study utilizing the Globus toolkit running on the GUSTO testbed.

1   Introduction

Parametric computational experiments are becoming increasingly important in science and engineering as a means of exploring the behavior of complex systems. For example, an engineer may explore the behaviour of a wing by running a computational model of the airfoil multiple times while varying key parameters such as angle of attack, air speed, etc. The results of these multiple experiments yield a picture of how the wing behaves in different parts of parametric space.

Over the past several years, we have developed a specialized parametric modeling system called Nimrod [1][2][3][17]. Nimrod uses a simple *declarative* parametric modeling language to express a parametric experiment and provides machinery that automates the task of formulating, running, monitoring, and collating the results from the multiple individual experiments. Equally important, Nimrod incorporates a distributed scheduling component that can manage the scheduling of individual experiments to idle computers in a local area network. Together, these features mean that even complex parametric experiments can be defined and run with little programmer effort. In many cases it is possible to establish a new experiment in minutes.

Nimrod has been applied to a range of application areas, including Bioinformatics, Operations Research, Network Simulation, Electronic CAD, Ecological Modelling and Business Process Simulation [14][17].

Whilst Nimrod has been very successful, it suffers from a few limitations when considered in the context of a "Global Computational Grid", in which a large number of computers are linked globally to form a seamless supercomputer [12]. First, it uses a static set of resources and does not "discover" new ones dynamically. This makes it more suited to fixed infrastructure than the type found on a global grid. Second, it has no idea of user deadlines. This is adequate when there are only a few users and a fixed resource, in which case, a user can estimate the time that a computation will take to complete, and adjust the experiment to meet project milestones. However, in a global grid, it is very difficult for the user to compute when a computation may complete because the underlying resources provide no performance guarantees, and Nimrod does not provide any mechanism to allow a user to specify such a deadline. Such deadline control, thus, becomes important as Nimrod is deployed in a global grid because the timeliness of a solution may be very important to the end user. Third, Nimrod relies on standard Unix level security. In a global grid, the owners of expensive supercomputing facilities require more elaborate security mechanisms. Finally, Nimrod does not support a range of access mechanisms, such as those provided by various queue managers. In the global grid many different queue managers co-exist and must be supported, since no single solution will be acceptable.

In this paper, we describe a new version of Nimrod, called Nimrod/G, which addresses these shortcomings. A particular focus of this paper is how the basic Nimrod model can be extended to provide (soft) performance guarantees in a dynamic and heterogeneous "Computational Grid". We

describe a simple but effective scheduling component in Nimrod/G which seeks to meet such constraints by a dynamic and iterative process of resource discovery, resource acquisition, and resource monitoring.

The paper introduces Nimrod and Globus, the toolkit on which Nimrod/G is built. It then describes the scheduling algorithm, and illustrates the utility of the system by a case study.

2 Parametric Modeling with Nimrod and Clustor

Nimrod is a tool that manages the execution of parametric studies across distributed computers. It takes responsibility for the overall management of an experiment, as well as the low-level issues of distributing files to remote systems, performing the remote computation and gathering the results. Clustor [4] is a commercial version of the research system Nimrod.

When a user describes an experiment to Nimrod, they develop a *declarative* "plan" file which describes the parameters, their default values, and the commands necessary for performing the work. The system then uses this information to transport the necessary files and schedule the work on the first available machine.

A plan file is composed of two main sections, the parameter section and the tasks section. Figure 1 shows a sample plan used for some of the simulation work discussed in this paper. The experiment consists of varying the thickness parameter and each execution receives a different seed value. Nimrod generates one job for each unique combination of the parameter values, by taking the cross product of all values. In the plan in Figure 1, 400 jobs would be generated to control the computational physics case study.

When the user invokes Nimrod on a workstation, the machine becomes known as the "root" machine because it controls the experiment. When the dispatcher executes code on remote platforms, each of these is known as a computational "node". Thus a given experiment can be conducted with one root and multiple nodes, each of a different architecture if required.

Nimrod supports five phases of a computational experiment. Phases 1 and 5 are performed once per experiment, while Phases 2, 3 and 4 are run for each distinct parameter set.

1. Experiment pre-processing, when data is set up for the experiment;

2. Execution pre-processing, when data is prepared for a particular execution;

3. Execution, when the program is executed for a given set of parameter values;

4. Execution post-processing, when data from a particular execution is reduced; and

5. Experiment post-processing, when results are processed, for example by running data interpretation or visualization software.

```
parameter iseed integer range from 100
to 4000 step 100;
parameter thick label "BUC thickness"
float range from 1.1
      to 2.0 step 0.1;
parameter jseed integer compute
thick*1000;

task nodestart
      copy ccal.$OS node:./ccal
      copy dummy node:.
      copy ccal.dat node:.
      copy skel.inp node:.
endtask
task main
      node:substitute skel.inp ccal.inp
      node:execute ./ccal
      copy node:ccal.op ccalout.$jobname
endtask
```

Figure 1 – Sample Plan file

During phases 2 and 4 files may be moved between the root machine and the cluster processes – unlike general parallel computing this is the only communication that occurs between tasks.

In this example there are two tasks – "main" and "nodestart". The "main" task is executed for each set of parameters. It runs a simulation, called ccal, on a node, passing the parameter values to the program via a parameter file. It then copies a number of result files back to the root machine, appending each name with a unique identifier. This task corresponds to the 3rd phase discussed in the previous section. The "nodestart" task executes once per experiment, and corresponds to the 1st phase described in the previous section. It copies a file to the remote node which is common across all simulations, and a skeleton of the input parameter file. The latter is processed using the substitute command, which replaces placeholders with actual values. It also copies the correct binary (ccal) for the target operating system.

The plan file is processed by a tool called the "generator". The generator takes the parameter values, and gives the user the choice of actual values. It then builds a "run" file, which contains a description for each job. The "run" file is processed by another tool called the "dispatcher", which is responsible for managing the computation across the nodes.

The dispatcher implements the file transfer commands, as well as the execution of the model on the remote node. In Nimrod and Clustor, the dispatcher allocates work to machines without any attempt to schedule their execution.

Nimrod/G, which is described in more detail in Section 4, makes use of two components of Clustor, namely the preparator and the generator. The dispatcher has been completely rewritten to support scheduling and make use of the Globus toolkit.

3 Globus

3.1 Toolkit functionality

The Globus Grid toolkit is a collection of software components designed to support the development of applications for high-performance distributed computing environments, or "Grids" [12]. The Globus toolkit is an implementation of a "bag of services" architecture, which provides application and tool developers not with a monolithic system but rather with a set of standalone services. Each Globus component provides a basic service, such as authentication, resource allocation, information, communication, fault detection, and remote data access [11][13][8]. Different applications and tools can combine these services in different ways to construct "Grid-enabled" systems. Four such components are used in Nimrod/G:

- The Globus Resource Allocation Manager (GRAM) service provides an API for requesting that computations be started on a computational resource, and for managing those computations once they are started.

- The Metacomputing Directory Service (MDS) provides an API for discovering the structure and state (e.g., availability) of resources that may be of interest to a computation.

- The Globus Security Infrastructure (GSI) provides single sign-on, run anywhere (where authorized) capabilities for computations such as ours that need to operate on many computers.

- The Global Access to Secondary Storage (GASS) service provides uniform access mechanisms and APIs for files stored on various storage systems.

The Globus resource management architecture is currently being extended to support advance reservations and to support differentiated services networks. Experiments with the advance reservation of differentiated services networks are being conducted in the context of both a local area testbed at Argonne and over an ESnet-based testbed linking ANL and LBNL. We anticipate this work being useful future versions of Nimrod, as a means of ensuring that computations can be performed on a required schedule.

3.2 Scheduling on the Grid

Scheduling programs on parallel and distributed computers has been the topic of research for many years, and a great deal has been accomplished. Much of the work has been targeted at scheduling and placing processes on a parallel computer in which the program can be represented as a task graph and the amount and nature of the computational resource is static for the duration of the execution. This environment leads naturally to scheduling heuristics, which try and produce a schedule that observes a number of constraints. Often there is a goal to minimise the execution time of the task or spread the load optimally.

The Computational Grid presents a much more challenging domain for scheduling because the performance of the underlying resource is highly variable, and can change even once an application has been scheduled. Accordingly, it is not possible to consider the Grid as a single computer system under the control of a single scheduler. The application domain of parameter studies also places additional demands on the scheduler. Such studies are often complex and very time consuming, and it is important to receive the results in a timely manner. Where traditional parallel computing scheduling systems may attempt to minimise the execution time of an application, or optimise the load distribution across the machines, we argue that allowing the user to specify an absolute (but soft) deadline is also a useful way of expressing the timeliness of the computation. Interestingly, deadlines have been commonly applied in real-time systems as the basis for scheduling.

Nimrod/G is a "Grid aware" application. It exploits an understanding of its problem domain as well as the nature of the computational Grid to provide a high level interface to the user. Specifically, it provides transparent access to the computational resources, and implements user level scheduling. In our case, we attempt to schedule otherwise unrelated tasks so that a user-supplied deadline is met. In the basic Nimrod model there is no communication between tasks once they have started, and thus the scheduling problem becomes one of finding suitable resources and executing the application. Although the problem sounds simple, the dimension of scheduling complexity is increased due to the introduction of parameters such as computational economics, deadlines, the usage of resources scattered over different administrative domains and varied machine and communication performance. A number of other researchers are also considering the scheduling problem in the context of their own applications, including Ninf [21], AppLeS project

[6][5], NetSolve [9], Nile [20] and NEOS [10]. In Condor [19] matchmaking is used to match resources with tasks, but parameter studies are not supported explicitly, as here, and no deadline scheduling is performed.

4 Nimrod/G Description

4.1 Architecture

Nimrod/G is designed to operate in an environment that comprises a set of sites, each providing access to a set of computers with their own administrative control, with access to these resources being mediated by a Globus Resource Allocation Manager (GRAM). Information about the physical characteristics and availability of these and other resources are available from the Globus directory service (MDS).

As illustrated in Figure 2, a user (or process acting on their behalf) initiates a parametric study at a *local site*; Nimrod/G then organizes the mapping of individual computations to appropriate *remote sites*, according to the Nimrod/G scheduling heuristic. On the local site, the *origin process* operates as the master for the whole system. The scheduling and monitoring logic described below is encapsulated in an *origin process*, which exists for the entire length of the experiment and is ultimately responsible for the execution of the experiment within the specified time and cost constraints. We note that this structure differs significantly from that used in Nimrod and Clustor as described in Section 2, because it has been designed to support the scheduling of computations on resources scattered across the globe with their own administrative policy and control.

The user is either a person or process responsible for the creation of the experiment. The user interacts with the origin process through a client process. Neither the user nor the client are required to be available all the time. The distinction between the Client and the Origin is useful because a client may be tied to a particular display or environment. The user can stop the client, move to another environment and start another client, without affecting the origin process and thus the progress of the experiment. In addition, it is possible for multiple clients to monitor the same experiment by connecting to the one Origin process. We have used this feature to allow a team in Melbourne and one in Chicago to observe the same computational experiment. A sample GUI for this monitor is shown in Figure 3. This example shows a number of jobs running on various GUSTO resources. It also shows the deadline and cost budget interface.

Each remote site consists of a *cluster* of computational nodes. A cluster may be a single multiprocessor machine, a cluster of workstations, or even a single processor. The defining characteristic of a cluster is that access to all nodes is provided through *a set of resource managers* provided by the Globus infrastructure. Each Globus Resource Allocation Manager (GRAM) implements a method for accessing processing resources. Typically, the method is a queue in a batch queuing system, such as Condor [18] or LSF [16] [23] or a process fork on a shared memory machine.
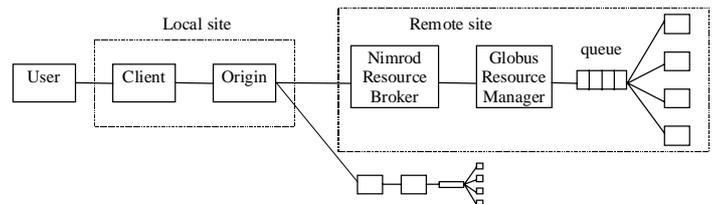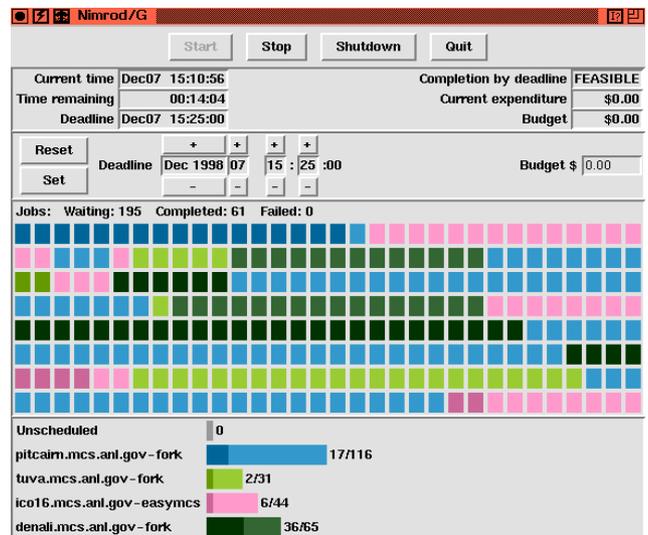

Figure 2 – Nimrod/G Architecture

Figure 3 – Sample monitor GUI

Before submitting any jobs to a cluster, the origin process uses the Globus process creation service to start a Nimrod Resource Broker (NRB) on the Cluster. The NRB is a different entity to the resource manager. It provides capabilities for file staging, creation of jobs from the generic experiment, and process control beyond that provided by the GRAM.

4.2 Cost in a Global Grid

Unless restrictions are placed on access to the various resources of a global grid, it is likely to become congested with too much work in short order. We believe a suitable model for controlling the amount of work requested is a fiscal one in which users pay for access. Such a scheme allows resource providers to set pricing rates for the various

machines, which can vary between classes of machine, times of the day, resource demand and classes of user. Even communication cost can be taken into consideration especially when data communication cost is significant compared to computation.

Whilst we have not developed a complete model of cost, and how it can be used in a global grid, we have experimented in Nimrod/G with defining a cost matrix between users and resources. For example, in Figure 4 the cost of resources 1 and 4 is different for users 1 and 4. In particular, user 1 finds resource 1 cheaper than 4, but user 4 find resource 4 cheaper than 1. This asymmetry means that the initial resources for each user may be different, providing for example a mechanism for favoring local machines first. At present we generate the cost matrix by hand based on a mutual agreement on the relative costs of systems. In the future, we expect that a more elaborate costing scheme will need to be devised, and this is a topic for ongoing research. In the next section we describe how cost is used in the current version of Nimrod/G.

Resources

		1	2	3	4	5
	1	1			3	
	2					
Users	3					
	4	2			1	
	5					

Figure 4 – Sample Nimrod/G Cost Matrix

4.3 Scheduling Algorithm

The Nimrod/G scheduler is responsible for discovering and allocating the resources required to complete an experiment, subject to specified execution time and budget constraints. This scheduling problem is made more complicated by the fact that in a Grid environment, we typically cannot guarantee exclusive or immediate access to underlying resources. Accordingly, *traditional* scheduling techniques (as discussed in [5]) are not necessarily applicable, as both resource availability and execution rates can vary unpredictably.

We address this problem by the use of the following simple heuristic:

1. (Discovery) First the number and then the identity of the lowest-cost set of resources able to meet the deadline are identified. A cost matrix (as described in Section 4.2) is used to identify low-cost resources; queries to the MDS directory service are then used to

determine resource. The output from this phase is a set of resources to which jobs should be submitted, ordered by the cost to *this* user. Different users may, therefore, generate lists sorted in a different order.

2. (Allocation) Unscheduled jobs (maintained in a pool, one per parameter set) are allocated to the candidate resources identified in Step 1.

3. (Monitoring) The completion time of submitted jobs is monitored, hence establishing an *execution rate* for each resource.

4. (Refinement) Rate information is used to update estimates of typical execution times on different resources and hence the expected completion time of the job. This refinement process may lead us to return to Steps 1 or 2 to discover new resources or to drop existing resources from the candidate set.

The scheme continues until the deadline is met, or the cost budget is exceeded. If the latter occurs, the user is advised and the deadline can be modified accordingly. One consequence on the way we have chosen to use cost is that the cost of an experiment will vary depending on the load and the profile of the users at that time. This reflects the type of pricing that occurs in an auction – less demand will allow the experiment to be performed on cheaper resources.

As we shall see below, initial experiments suggest that this heuristic works well in practice: applications tend to determine quite quickly what is an appropriate job submission rate, and use expensive resources only when necessary.

5   A Case Study

We report on experiment that we have conducted to evaluate the effectiveness of the Nimrod/G architecture and scheduling heuristics in a real application. The experiment was using resources provided by sites participating in the international Globus Ubiquitous Supercomputing Testbed Organization (GUSTO). GUSTO sites run standard Globus software and can sometimes be persuaded to make their resources available for Grid computing experiments. As show in Table 1, the study reported here used resources at Argonne, Boston University, Monash University, Northern Illinois University, USC Information Sciences Institute, and the University of Wisconsin. These resources are quite diverse in terms of their size, availability, architecture, processing capability, power, performance, scheduling mechanism (immediate access or "fork", Condor-based queue, LSF-based queue), and geographic location. The "Max number nodes" column in Table 1 indicates the number of nodes in the machine, while "Average free nodes"

indicates the average number available during the experimental period in April 1999. The table also shows a (currently artificial) cost value for each system.

5.1 Ionization Chamber Calibration

The Australian Radiation Protection and Nuclear Safety Agency (ARPANSA) provides the Primary Australian Standards for certain Radiological quantities such as Air Kerma, which essentially quantifies photon energy deposition in terms of ionization in air. The calibration of medical equipment used for diagnosis and/or treatment against these standards ensures that in any location the radiation doses received by patients is consistent with that received in other locations in Australia and overseas.

An ionization chamber essentially isolates a certain volume of air and measures the ionization within that volume. However, the physical process of isolating a volume of air modifies the original photon and electron spectrum entering the volume. Thus, if the chamber is to act as a primary standard for calibration purposes, it is necessary to correct the measured ionization for the spectral perturbations due to the physical presence of the chamber.

The nature of the problem is such that not all correction factors can be determined experimentally. Within the field of radiation dosimetry, Monte-Carlo simulation programs, such as the EGS4 package [22], have become an acceptable method of simulating the response of ionization chambers to photons and electrons and thus an alternative means of determining the appropriate correction factors. The absolute accuracy of the EGS4 package for simulating ion chambers is quoted at about 1% but relative accuracy is actually better than this.

One important correction factor for the primary standard, in the standardization of photons emitted by a cobalt-60 teletherapy source, is the extent of photon attenuation and scattering in the front wall of the chamber. Detailed measurements of chamber response as a function of front wall thickness have been performed and a quantitative model of interactions within the front wall used to determine the appropriate correction factors. These detailed measurements provide an interesting basis for comparison with Monte-Carlo, as not only the chamber but also the cobalt-60 photon spectrum are to be simulated. The calculations reported here concern the simulation of the chamber response as a function of front wall thickness. We used Nimrod/G to perform this parametric variation whilst also running the model many times with different random number seeds and different front wall thicknesses. As the calculated data may not be normally distributed, it was necessary to calculate the averages of sets of data and to take the mean of the averages to obtain a mean value (whose components are normally distributed) for any particular front wall thickness.

5.2 Scientific Results

One of the most important output variables of the model is the number of ion pairs created in the collecting volume per incident photon. Accordingly, we wish to compare the simulation data with experimental data. In order to do this, we fit both the experimental and simulation results to an expression of the form $Ae^{-\mu t} + Be^{-\beta t}$, where μ is the effective attenuation coefficient for photons, β is the effective attenuation coefficient for electrons and t is the thickness of the front wall of the chamber. A comparison for the fitted coefficients for both the simulation and experiment is given in Table 2. These results show that the EGS4 model is quite accurate. Further, by considering the data as plotted against thickness in Graph 1, the model is particularly accurate when the thickness is less than 0.4 cm.

In order to improve the accuracy of the model, we believe that we require in the order of 5 times the number of simulations. Given that the results could be obtained on a relatively modest number of processors in the order of a working day, this should be achievable easily. We plan to perform these simulations in the near future.

5.3 Computational Results

The simulations described were run on the subset of the GUSTO testbed resources defined in Table 1. Of the 364 nodes in this subset, about 70 were available to us during the trial. This number was sufficient to allow testing of many of the key features of Nimrod/G, in particular its ability to schedule tasks according to time and cost constraints.

The ionization chamber study involved 400 tasks. The execution time of the model varied depending on the platform used, ranging from about 45 minutes per parameter set on the SGI R10000 based machines to 140 minutes on the NIU and ISI Suns. Three separate experiments were performed with deadlines of 20 hours, 15 hours, and 10 hours, respectively, so as to allow an evaluation of Nimrod/G's ability to meet soft real time deadlines. Graph 2 shows the number of nodes used as a function of time for each deadline. Not surprisingly, Nimrod/G allocates additional resources for the more stringent deadlines. Whilst this result appears obvious, it should be stressed that these results have been obtained from a *real* test-bed, in which the resources are *discovered* and the load has been distributed *dynamically*. The algorithm has adapted the distribution of tasks without any prior knowledge of the initial load, test-bed configuration or speed of the individual machines.

Machine	Loca-tion	Method of starting Jobs/ Machine Type	Rel. Cost	Max nodes	Typ. avail
Pitcairn	ANL Chicago	Fork UltraSPARC-II	5	8	3
Tuva	ANL Chicago	Fork UltraSPARC-II	5	1	1
Goshen	U. Wisco	Condor Sun 4	5	175	7
Lemon	ANL Chicago	Fork MIPS R10000	10	2	2
Yukon	ANL Chicago	Fork MIPS R10000	10	16	16
Flash	ISI LA	Fork MIPS R10000	15	2	2
Jupiter	ISI LA	Fork MIPS R10000	15	8	8
Olympus	North Illinois	Fork UltraSPARC	15	1	1
Bolas	ISI LA	Fork UltraSPARC	20	1	1
Hammie	ISI LA	Fork UltraSPARC	20	1	1
Huntsman	ISI LA	Fork UltraSPARC	20	1	1
Hathor	Monash Australia	Condor Pentium II/333	50	20	1
Denali	ANL Chicago	Fork MIPS R10000	50	96	24
Lego	Boston U	LSF MIPS R10000	50	32	6
Total				364	70

Table 1 – GUSTO machines used for case studies.

parameter	EGS4 calculation	experiment
μ	0.103	0.113
β	26.61	29.90
Ratio (B/A)	0.377	0.338

Table 2- Attentuation coefficients

Of more interest are Graphs 3, 4 and 5, which break down selected nodes according to cost, against time. These graphs show that the scheduler initially favours cheaper resources, but then, as time advances and it finds that the deadline cannot be achieved using the current resource set, more expensive machines are introduced. For example, in Graph 3 the 10 "cost unit" machines are introduced after about half an hour when the scheduler calculates that it cannot meet the 20 hour deadline using only 5 "cost unit" machines. Similarly, in Graph 5, it can be seen that 50 "cost unit" machines are introduced at around 2 hours in order to meet the 10 hour deadline, whilst these were not required in the 20 hour experiment. These predictions are based on the

measurements of the real performance of the machines whilst running the ionization code.

Table 3 quantifies the impact on cost of the different node selections made for different deadlines: a 10 hour deadline costs more than three times a 30 hour deadline. This is a reflection of the distribution of resources available at the time we performed the work, and the particular cost matrix which was used. While we cannot easily show in such a dynamic environment that Nimrod/G is making "optimal" selections, it is clear that Nimrod/G is being effective in selecting more expensive nodes only when the system requires them to meet the deadline. Of course, these costs will vary depending on the current load on the test-bed as the lowest cost resources may not always be available.

Experiment	Total Cost in Cost Units
10 hours	9060
15 hours	5675
20 hours	2980

Table 3 – Cost of experiments

6   Conclusion

In this paper we have discussed the evolution of a particular tool, Nimrod, from a local computing environment to the Global Computational Grid. The paper has described the various services of Globus and how these have been applied in building a Grid aware application. In particular, we have focussed on the problem of providing some soft real time deadlines in such an environment. The algorithm used is both simple and adaptive to changes in the workload distribution on the grid, and incorporates user requirements as well as system ones. The case study has illustrated that it is possible to build an application which takes account of the highly dynamic and unpredictable nature of the grid.

However, we regard our efforts in this area as preliminary. We need to develop a more complete model of cost in such an environment, in particular issues such as storage of money and how it can be deployed. We wish to take account of the ability in Globus to reserve some resources, and incorporate this into the scheduling mechanism. It would appear that some notion of privilege or priority in addition to money would be important, particularly when a user wishes to make use of local resources, but also place them in the global grid. For example, under the current scheduling algorithm, it is possible for a user to be forced to use external, more expensive, resources just because the local ones are already working for another user. Finally, there are a number of implementation issues which need to be addressed, such as the way that Nimrod/G explores the Globus MDS in search of suitable machines. Such issues are the topic of ongoing research.
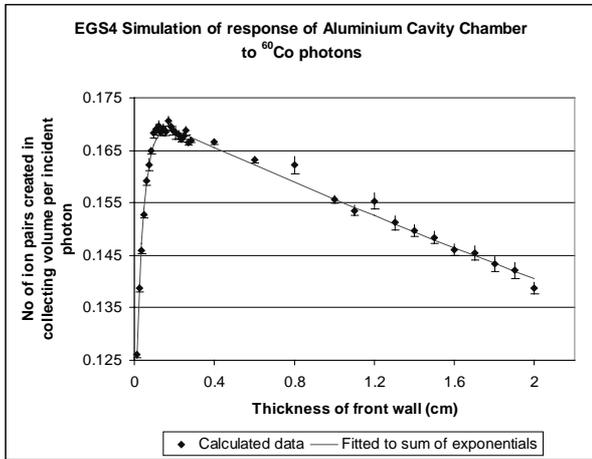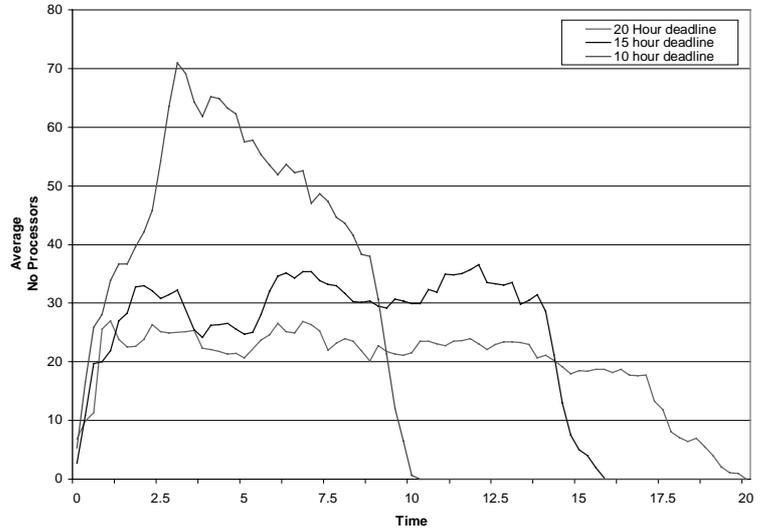
Acknowledgements

References

[1] Abramson , D., Sosic , R., Giddy , J., Cope , M. "The Laboratory Bench: Distributed Computing for Parametised Simulations", 1994 Parallel Computing and Transputers Conference, Wollongong, Nov 94, pp 17 - 27.

[2] Abramson D., Sosic R., Giddy J. and Hall B., "Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations", The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.

[3] Abramson, D., Foster, I., Giddy, J., Lewis, A., Sosic, R., Sutherst, R. and White, N. "Nimrod Computational Workbench: A Case Study in Desktop Metacomputing", Australian Computer Science Conference (ACSC 97), Macquarie University, Sydney, Feb 1997.

[4] Active Tools Corporation, http://www.activetools.com

[5] Berman, F. "High Performance Schedulers", In The Grid: Blueprint for a Future Computing Infrastructure, pages 279 - 309. Morgan Kaufmann Publishers, 1999

[6] Berman, F. and Wolski, R. "The AppLeS project: A status report", In proc. NEC Symposium on Metacomputing, 1997.

[7] Bestavros, A. "Load profiling: A methodology for scheduling real-time tasks in a distributed system." In Proceedings of ICDCS'97: The IEEE International Conference on Distributed Computing Systems, Baltimore, Maryland, May 1997.

[8] Bester, J., Foster, I., Kesselman, C., Tedesco, J. and Tuecke, S. "GASS: A Data Movement and Access Service for Wide Area Computing Systems", Proc. IOPADS'99, ACM Press, 1999.

[9] Casanova, H. and Dongarra, J. "NetSolve: A Network Server for Solving Computational Science Problems", The International Journal of Supercomputing Applications and High Performance Computing, Vol 11, Number 3, pp 212-223, 1997.

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

[11] Fitzgerald, S., Foster, I., Kesselman, C., von Laszewski, G., Smith, W. and Tuecke, S. "A directory service for configuring high-performance distributed computations.", Proceedings 6th IEEE Symposium on High Performance Distributed Computing, , 365-375, IEEE Press, 1997.

[12] Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. In The Grid: Blueprint for a Future Computing Infrastructure, pages 259-278. Morgan Kaufmann Publishers, 1999.

[13] Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. "A Security Architecture for Computational Grids", ACM Conference on Computers and Security, 83-91, ACM Press, 1998.

[14] http://hathor.cs.monash.edu.au/RJK

[15] http://www.globus.org/

[16] http://www.platform.com

[17] Lewis, A., Abramson D., Sosic R., Giddy J., "Tool-based Parameterisation : An Application Perspective", Computational Techniques and Applications Conference, Melbourne, July 1995.

[18] Litzkow, M., Livny, M. and Mutka, M.W., "Condor – A Hunter of Idle Workstations", proceedings of the 8th International Conference of Distributed Computing Systems, pp 104-111, June 1988.

[19] Livny, M. High-Throughput Resource Management. In The Grid: Blueprint for a Future Computing Infrastructure, pages 311-337. Morgan Kaufmann Publishers, 1999

[20] Marzullo, K., Ogg, M., Ricciardi, A., Amoroso, A., Calkins, F. and Rothfus, E. "NILE: Wide area computing for high energy physics", proc. 1996 SIGOPS Conf. New York, ACM, 1996.

[21] Matsuoka, A., Nagashima, U. and Nakada, H. "Ninf: Network based information library for globally high performance computing", in Methods and Applications (POOMA), 1996.

[22] Nelson, W.R., Hirayama H. and Rogers, D.W.O. "The EGS4 Code System", Stanford Linear Accelerator Center Report SLAC-265 (1985)

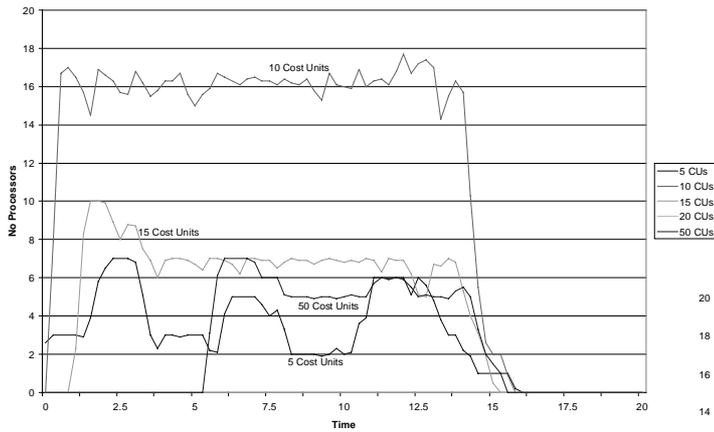[23] Zhou, S., Zheng, X., Wang, J. and Delisle, P. "Utopia: A load sharing facility for large,

heterogeneous distributed computer systems", Software-Practice and Experience, 23(12): 1305- 1336, December 1993.

Graph 1 – Chamber response against thickness

EGS4 Simulation of response of Aluminium Cavity Chamber to ^{60}Co photons

No of ion pairs created in collecting volume per incident photon

Thickness of front wall (cm)

◆ Calculated data ——— Fitted to sum of exponentials

Graph 2 - GUSTO Usage for Ionization Chamber Study

Average No Processors

Time

20 Hour deadline
15 hour deadline
10 hour deadline

Graph 4 - GUSTO Usage for 15 Hour Deadline

No Processors

Time

10 Cost Units
15 Cost Units
50 Cost Units
5 Cost Units

5 CUs
10 CUs
15 CUs
20 CUs
50 CUs

Graph 3 - GUSTO Usage for 20 Hour Deadline

No Processors

Time

10 Cost Units
5 Cost Units

5 CUs
10 CUs
15 CUs
20 CUs
50 CUs

Graph 5 - GUSTO Usage for 10 Hour Deadline

No Processes

Time

50 Cost Units
10 Cost Units
15 Cost Units
5 Cost Units
20 Cost Units

5 CUs
10 CUs
15 CUs
20 CUs
50 CUs