TRANSFER THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Probabilistic Grid Scheduling

Based on Job Statistics and Monitoring Information

Aleksandar Lazarevic

Department of Electronic and Electrical Engineering
University College London
London, England 2005

Supervisor: **Dr. Lionel Sacks**

Words: 27729

Table of Contents

Abstract

This transfer thesis presents a novel, probabilistic approach to scheduling applications on computational Grids based on their historical behaviour, current state of the Grid and predictions of the future execution times and resource utilisation of such applications. The work lays a foundation for enabling a more intuitive, user-friendly and effective scheduling technique termed *deadline scheduling.*

Initial work has established motivation and requirements for a more efficient Grid scheduler, able to adaptively handle dynamic nature of the Grid resources and submitted workload. Preliminary scheduler research identified the need for a detailed monitoring of Grid resources on the process level, and for a tool to simulate non-deterministic behaviour and statistical properties of Grid applications.

A simulation tool, GridLoader, has been developed to enable modelling of application loads similar to a number of typical Grid applications. GridLoader is able to simulate CPU utilisation, memory allocation and network transfers according to limits set through command line parameters or a configuration file. Its specific strength is in achieving set resource utilisation targets in a probabilistic manner, thus creating a dynamic environment, suitable for testing the scheduler's adaptability and its prediction algorithm.

To enable highly granular monitoring of Grid applications, a monitoring framework based on the Ganglia Toolkit was developed and tested. The suite is able to collect resource usage information of individual Grid applications, integrate it into standard XML based information flow, provide visualisation through a Web portal, and export data into a format suitable for off-line analysis.

The thesis also presents initial investigation of the utilisation of University College London Central Computing Cluster facility running Sun Grid Engine middleware. Feasibility of basic prediction concepts based on the historical information and process meta-data have been successfully established and possible scheduling improvements using such predictions identified.

The thesis is structured as follows: Section 1 introduces Grid computing and its major concepts; Section 2 presents open research issues and specific focus of the author's research; Section 3 gives a survey of the related literature, schedulers, monitoring tools and simulation packages; Section 4 presents the platform for author's work – the Self-Organising Grid Resource management project; Sections 5 and 6 give detailed accounts of the monitoring framework and simulation tool developed; Section 7 presents the initial data analysis while Section 8.4 concludes the thesis with appendices and references.

Probabilistic Grid Scheduling

Department of Electronic and Electrical Engineering
University College London
London, WC1E 7JE
U.K.
a.lazarevic@ee.ucl.ac.uk
www.ee.ucl.ac.uk/~alazarev/

1. CHAPTER ONE

INTRODUCTION TO THE GRID

The evolution of distributed computing, ultimately leading to the emergence of the Grid paradigm, was set in motion in 1994 with the start of the Legion and Globus projects.

This chapter gives a brief introduction to Grid computing with Section 1.1 arguing the case for distributive computing; Section 1.2 outlining the unique and novel aspects of the Grid approach; Section 1.3 giving further details on the Globus Toolkit, a *de facto* Grid middleware today. Section 1.4 introduces organisations and projects closely supporting Grid efforts, while Section 1.5 concludes the chapter.

1.1. A Case for Distribution

A long running battle in the evolution of computer architectures is that between centralised and distributed approaches. In last fifty years, technological advances often tilted the battle in favour of the centralised paradigm, always providing means to concentrate more computational power into smaller, more integrated space. The distributed camp, on the other hand, was stimulated by the developments of new, computationally and data challenging applications, constantly being one step ahead of the resources any centralised installation could provide. It has therefore been natural that the development of both approaches would continue in parallel, each finding an application space that it can serve best.

However, social, political and scientific developments in the last decade provided a strong spring board for the distributed computing paradigm. Following, or perhaps leading, international integration efforts, scientific research has moved

from closed university campuses and governmental departments into a more cross-border collaboration effort, spanning many countries, organisations and funding bodies. Stable political climate resulted in the willingness of funding bodies to invest money in non-dedicated facilities, open for use by scientist from other nations. And with research goals becoming ever more challenging, a substantial shift into "Big Science" was unavoidable – requiring enormous investment in infrastructure and research facilities often beyond the reach of even the most developed nations.

The same decade saw a number of advances in information technologies able to support distributed computing better than ever before. High speed networking became ubiquitous with the roll out of extensive optical networks, effectively rendering bandwidth a linear function of (moderate) monthly expenditure. IP protocol established itself as a standard for all and any kind of networking, with universal connectivity slowly becoming a reality. Despite some pessimistic views, Moore's Law still held firm ground, becoming applicable not only to silicone chips, but to magnetic storage products as well. Storage space became a commodity few people saw limits to, also with a linear function of unit cost. Despite all these advances, and frequent promises by the industry, one problem still caused headaches in the high performance computing circles, marred deployment of even bigger data centres, and played nicely into hands of distributed computing advocates. Power consumption of silicone chips remained directly proportional to their computational output throughout the decade.

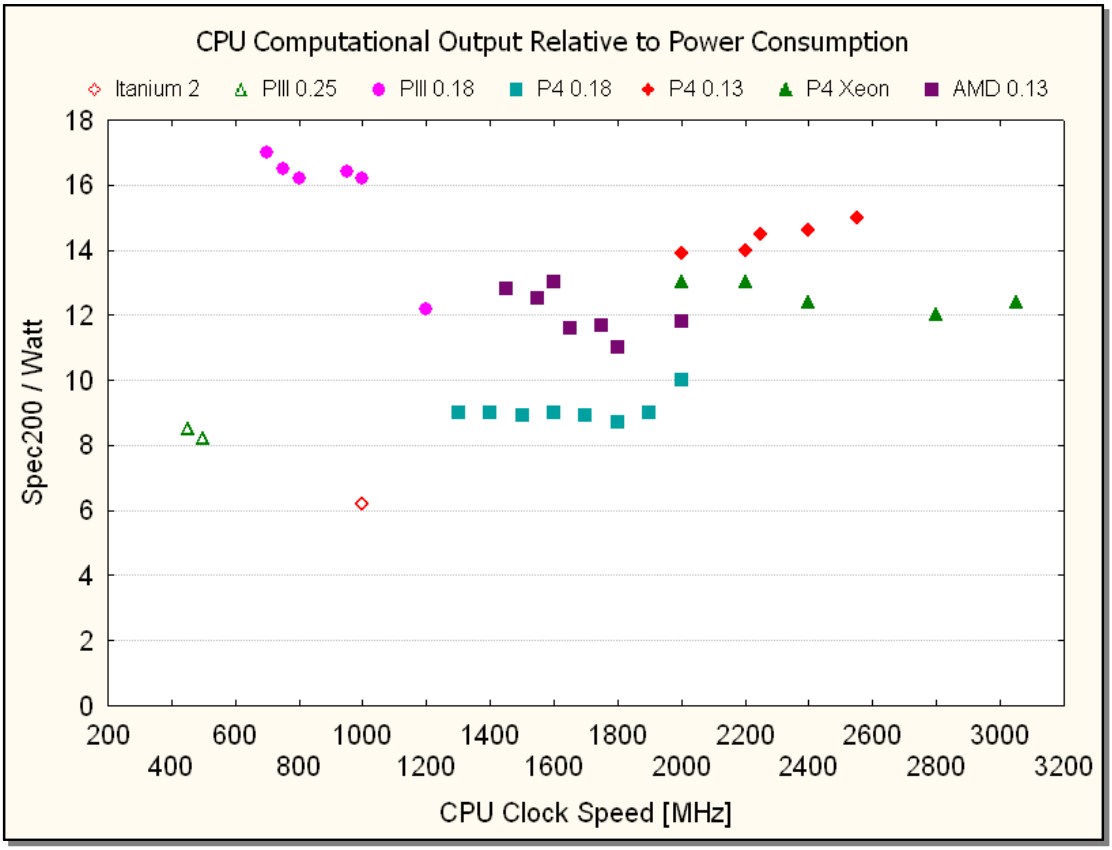


Figure 1 CPU Power Consumption Relative to SPEC Computational Output [1]

Figure 1 shows the computational output per watt of consumed electricity for a range of CPU families at different operating frequencies.

Hard disks and other components are consuming more power than ever. High levels of integration could not increase the computational density of data centres any longer as physical limits are being reached with respect to power supply, distribution, and heat dissipation (air-conditioning). Although ways of overcoming these through the use of "brute force" exist, expense of such approaches would significantly impact the price per FLOPS ratio.

At the turn of the century, right set of enabling technologies and target markets has formed inspiring new enthusiasm for distributed computing approaches.

1.2. The Grid Perspective

Distributed computing was certainly not a new phenomenon in the high performance computing field. Indeed, distributed computer installations were both researched and deployed throughout the 1980's with Digital Equipment Manufacturer (DEC) leading the way with its super-mini clusters based on DECnet. These early attempts were often less then the sum of their parts, requiring great effort to set- configure hardware and software, they used proprietary protocols and behaved more like a monolithic unit sliced and arranged over some physical distance.

Although similar ideas have been floating in the research community for years, the first prominent project which popularised distributed computing was SETI@Home[2]. By running as a screensaver on Internet connected PCs, SETI@Home utilised their idle time to comb radio signals for signs of extraterrestrial life. A whole range of applications, aimed at utilising unused cycles on distributed workstations, developed from this early attempt and the approach was coined "cycle scavenging". Success of SETI@Home spawned a number of similar derivative projects (like Stanford University Folding@Home[3] protein folding application), but by far the most successful is the Condor Project (see Section 3.2.2).

Grid computing[4] drew significant inspiration from the power grids in which electricity generation is remote from the point of consumption, and transparent to the user. From the user's perspective, the power grid offers ubiquitous and reliable access to virtually unlimited amount of electricity on a pay-per-use model. The goal of providing computational power as a form of utility thus became an epitome of Grid Computing.

The Grid computing perspective was to capitalise on the benefits of distributed architectures while working on the major problems and issues uncovered in previous attempts. While many implementations of Grid concept exist, varying greatly in form and function, the primary objective was to develop a transparent platform, based on industry standard protocols and open source code, which can be ported to any of the many operating systems and architectures currently in

use by the academic and commercial institutions. The Grid was to be a much more organic network than its distributed predecessors, able to form on-demand Virtual Organisations (VO) [5] spawning geographical, networking and administrative boundaries. Grid middleware would be able to integrate distributed computational, storage and visualisation resources into persistent environments, provide a strong security layer, and a resilient, burstable platform for scientific research and commercial applications. In an ideal scenario, Grids would become a new, pervasive and transparent utility which is autonomous and self-manageable.

1.3. The Globus Toolkit and its Impact

Globus Toolkit[6] has emerged as a *de-facto* standard Grid middleware. Initially developed by Ian Foster, Steve Tuecke and Carl Kesselman, Globus now enjoys a large research and development community compromising universities, standards bodies and large international corporations. The toolkit has gone through three major versions, with the version 2.4 widely accepted as most stable, and extensively deployed in the academic community. Although Globus Toolkit has been used in production environments (substantially modified and repackaged), it is still primarily a research framework for the Grid concept. Commercialisation and production of a industry-grade platform is being actively pursued by Globus Alliance members.

Version 3 of Globus Toolkit embraces the services framework based on the Web Services Resource Framework (WSRF), and so will the recently announced Version 4. Regardless of the implementation issues, Globus Toolkit effectively stands on three pillars: Globus Security Infrastructure (GSI), Globus Resource Allocation Manager (GRAM) and Monitoring & Discovery Service (MDS).

Globus Security Infrastructure is based on public key concepts (PKI) and X.509 certificates. Each Globus-enabled network host, service, or user has a certificate which is used in authenticating that entity's identity and authorising access to a resource. All messages communicated between Globus-enabled nodes or components are also secured using Transport Layer Security (TLS) with corresponding certificates. The security that GSI is able to offer using PKI technology is widely recognised, although delivering such levels of security has proven to be one of the biggest problems with Globus Toolkits. Public key infrastructure was never trialled on such a large scale as commanded by the wide acceptance of Globus, and many issues concerning the set-up and running of Certification Authorities, maintenance of certificate revocation lists, and user's approach to dealing with certificates have since arisen. These are not necessarily faults with the PKI technology or the Globus toolkit, but rather point to a changing security landscape, and prompt a more coordinated action for developing supporting tools and altering users habits and perceptions.

Globus Resource Allocation Manager could be thought of as the core of the toolkit, an interface common to all nodes in a Globus Grid. On a network facing

side, GRAM answers queries from other Globus nodes for running applications on the local system, providing those requestors with a unique resource identifier (URI) contact string which can be used at a later time to query the progress of the job, and collect the job's output. On the local system side, GRAM can interface any number of local schedulers, from simple UNIX *fork* to systems such as Portable Batch System (PBS, see section 3.2.5) or Load Sharing Facility (LSF, see Section 3.2.6), through a modular shell script. Once the job is submitted, GRAM captures its standard and error outputs, and provides monitoring facility for proper/improper termination.

Monitoring & Discovery Service is a Lightweight Directory Access Protocol (LDAP) server based component that keeps information pertaining to the current state of a Globus host and its hardware capability and software environment. MDS is built on a hierarchical model, with individual information providers reporting single metric measurements to a tree of distributed information service servers (called GRIS and GIIS). MDS is a central point of contact for locating resources, obtaining their usage statistics and discovering services they are able to provide. For such a critical mission, MDS's performance leaves plenty to be desired. The move to web services platform will see important changes in the information service aspect.

1.4. Wider Grid Landscape

Globus Toolkit was not the first Grid middleware to emerge from the academic institutions. The Legion Project (see Section 3.2.10), started in the late 1993 at the University of Virginia had many similarities with the later Globus project. However, Legion did not manage to solicit the same amount of interest and involvement as Globus, and although it remains an ongoing research effort, the installed base remains small.

The success and popularisation of Globus Toolkit has provided a focus point for Grid research, while the Globus Alliance has acted as a strong promoter of Grid computing in international research, academic and commercial bodies. Globus Toolkit was used as the base of several derivative middleware currently in production use in large international projects (EGEE[7], DataGrid). A large user community has developed and is orchestrated through the Global Grid Forum (GGF) [8], a regular meeting of formal research and working groups tackling short- and long-term issues of Grid Computing. Commercial Grid implementations were supported by a number of university spin-offs formed by researchers on Grid related projects. Companies like Platform Computing[9], Avaki[10] and United Devices[11] managed to establish themselves as leading consultancies in the field. Major market players have in the last three years recognised the marketing potential of the Grid, and have joined GGF as industrial collaborators and development partners. IBM, Sun Microsystems and Hewlett-Packard are all offering Grid solutions for the enterprise. It is no surprise that the number of commercial, production level Grid deployments is constantly on the rise, and that the Grid was named as one of top ten

technologies that could change the word in the next decade[12]. The academic community equally benefited from this surge of interest in Grid computing with the increase in research grant funds and better visibility of their work. But with research proposals influenced by political agendas and policy makers, a negative effect may be caused by overselling the Grid's near-term potential, and not fulfilling community's (over-optimistic) expectations.

1.5. Conclusions

In this chapter, the need for distributed computing, and some major driving factors of its development have been discussed. An overall perspective of the Grid concept was given, and its main value proposition was underlined. The Globus Toolkit was introduced and its main components briefly discussed. Finally, an overview of other Grid related efforts is given and major commercial supporters identified.

2. CHAPTER TWO

RESEARCH AREA

Although wide-area Grid computing can be seen as a natural step from the distributed computing concept, it poses many new challenges and requires significantly different implementation approaches. In this chapter, areas of major Grid research will be presented, and research challenges of the author's current and future work described in more detail.

Section 2.1 introduces Grid middleware components and crucial improvements needed; Section 2.2 gives the main focus of the author's work, while Sections 2.3 and 2.4 respectively present issues in simulating Grid environments and monitoring Grid resources.

2.1. Open Issues in Grid Computing

Various aspects of the distributed computing paradigm have been researched throughout the 1980s and 90s, with solid, proven solutions for many implementation and programming challenges. Despite all its similarities and common roots to legacy distributed computing, the Grid poses radical new questions and requires new approaches for solving them. The Grid's added value proposition is in supplying computing power as a utility, providing an ubiquitous on-demand service through a semi-persistent environment created for solving a specific task (Virtual Organisation). This is in complete contrast with legacy cluster systems, and their strict plan-deploy-use cycle. Therefore, legacy approaches and solutions can not simply be migrated onto the Grid middleware, as they would diminish the core benefit this new technology offers.

Large amount of Grid research is being undertaken in all aspects of Grid middleware: data management, security, networking, scheduling and resource

management. The Grid's envisaged ubiquity and flexibility – ability to operate on any (time and space shared) hardware, interconnected by any network (dedicated or not), and deployable across administrative boundaries – adds a whole new layer of complexity to legacy distributed computing problems. It follows that in developing core Grid middleware components one should assume little of the operational environment, and require even less, aiming for an adaptable system able to operate in a wide range of conditions.

Creating a global and dynamic computational network also creates considerable management problem. After the initial research effort to develop and deploy world's first Grid services, the problem of managing systems of such global scale became more prominent. The large management burden is caused by the scale and heterogeneity of the platform, outdated management tools, and the reluctance to radically change management practices. Desirable properties of any new or improved Grid middleware components would hence be a high degree of autonomy and self-management, and a low impact on end users and their workflow.

The Grid middleware will be greatly influenced by the nature of the applications that run on it. The Grid has already enabled scientific simulations and experiments to be performed at previously impossible scale, but as it becomes a widely accepted collaborative computing platform the application set is likely to change. With development of computational markets, users could find it cheaper and more convenient to use the Grid for an increasing variety of jobs. The grid may emerge as a generalised service delivery platform, executing large numbers of medium and low demand computational jobs. This would lead to a shift from a few highly specialised and demanding applications to a more diverse application landscape.

Any such changes in the usage profiles would change a number of important job statistics which current management components rely on. As the applications execution times fall, job arrival rates will increase, and so will the resource discovery and scheduling overheads. Current Grid resource discovery and scheduling components are built on assumptions of a very long execution times and resource pools of modest size. Overheads and job submission delays now introduced by the Grid middleware may be considered insignificant, but in the future may represent the greatest part of the job execution time. In a general use case, schedulers will have to make an intelligent decision and adjust the complexity of resource discovery and scheduling to the likely complexity of the job at hand.

2.2. Research focus: Grid Scheduling

The author's main focus is research and development of a flexible scheduling system, better suited to the user's normal workflow, and enabling higher utilisation of Grid resources. Most of the current Grid schedulers offer either a "fair-share" of resources to all users, or apply fixed parameter modifiers based on

the user or group priorities. These schedulers are predominantly batch based systems, requiring users to submit jobs to queues with resource utilisation caps, and the administrators to prioritise queues and users using scheduling policies. The end effect is underutilisation due to idle periods, or lower than expected quality of service to the users whose jobs fail to capture required share of the resources.

The author's view is that better scheduling can be achieved by enabling the user to specify an intuitive metric, such as job completion deadline or available "budget", and producing a schedule optimising these metrics. These metrics are embedded in the way users commission services in the real world – asking for them to be delivered in certain time and at a certain cost. The notion of deadline based scheduling is the central and overriding motivation of the research work herein presented.

The problem of deadline scheduling reduces to the prediction of the execution time of any given application submitted to the scheduler. Should this be known, admission decision can be made based on the current load of the system and user's requested turnaround time. A real-time scheduling plan could then be made using any number of optimisation approaches – a Tetris™ game of fitting blocks of jobs to a timeline. It is clear that the knowledge of how long a certain job will take to complete is the crucial information needed to support deadline scheduling. A difficult task on its own, this information should be arrived at with a limited system overhead, in a short period of time, and with reasonable accuracy. Such deadline driven scheduler must be able to quantify the quality of its predictions, and be able to take adaptive actions in case a wrong prediction disturbs the execution plan. And with the user convenience as one of the main requirements, switching cost and increase in job submission complexity should be minimal.

2.3. Simulation and Testing

Redesigning a crucial element of a system, such as the scheduler, raises problems in its testing and quality assessment. Deploying an unproven and possibly unstable scheduler on a production system is not acceptable. However, assessing the performance and benefits of the new scheduler may be impossible in a simulation environment in which many crucial metrics are predetermined and static. Any results and findings arrived at purely through simulations may hold little credibility, and may be insufficient to support deployment of the scheduler on the production system.

The need for an adequate tool for simulating real Grid application load was recognised early on in this research. The aim was to develop an application able to stress the scheduler, and all other components of the Grid middleware, running on a hardware testbed. The computational load presented to the Grid should have properties similar to the ones observed on a production system, be

repeatable and deterministic on large scale, yet probabilistic on lower scales to allow interesting usage patterns to develop.

Thus, a secondary line of research should investigate and develop appropriate supporting tools for simulating real scheduling problems and testing developed solutions in an environment closely resembling production Grids.

2.4. Measurements & Information Flow

Regardless of a method adopted for delivering deadline scheduling, measurements of system's historical performance, and a timely and accurate snapshot of current activities is essential. Monitoring of the Grid is difficult due to the heterogeneous nature and large number of resources that need to be observed. Monitoring systems with predefined sampling points and frequencies will inevitably end up with poor information capture – high volumes of irrelevant measurements in which a truly important observation may be lost. Operating in a geographically distributed environment, transferring monitoring information indiscriminately leads to inefficient use of bandwidth. The next generation of truly effective Grid monitoring systems would have to be more intelligent, flexible and agile, adapting the granularity, frequency and the communication methods to the state of the operating environment and the importance of the measurements. These systems would not be unlike virtual sensor network, permeating the Grid fabric and self-organising in monitoring constellations according to the current requirements.

Monitoring systems currently in use on the Grid (to be discussed in Section 3.3) measure the total CPU load on a single host. However, one can not assume this figure is the equivalent of the CPU utilisation of the application running on that node. Grid hosts would generally be both time- and space-shared with other Grid users, and even with other, local users. Furthermore, author's tests of the Grid middleware deployed on a small testbed (see Figure 4) showed that a significant computational overhead is evident in all phases from job submission to job results collection.

It was therefore deemed necessary for a more granular monitoring system to be developed, one able to monitor CPU utilisation of a specific application or a process. This information would have to be sampled with sufficient frequency to extract statistical features of the processes and be easily accessible by other Grid management components.

Investigation into current monitoring tools, and conceptual development of a suitable monitoring framework will be undertaken as a secondary line of research. The focus will be on demonstrating the benefits of an integrated monitoring-scheduling-accounting approach, rather than delivering a completely new monitoring tool.

2.5. Conclusions

Chapter 2 offers a view on the differences between legacy distributed systems and the Grid, identifying new challenges and suggesting novel approaches that will be required for solving them. Scheduling of computational jobs on the Grid is identified as the main focus of author's current and future work. Simulation of realistic Grid environments and highly granular monitoring are seen as essential requirements for successful study of advanced scheduling algorithms and will be pursued as secondary research objectives.

3. CHAPTER THREE

LITERATURE SURVEY

Despite being a relatively new research topic, Grid computing has already attracted attention of many scientists, research groups and standards organisations. Previous work in the area of parallel and distributed computing provides a firm foundation and is sill largely applicable to the Grid. However, resource management, and particularly scheduling, require innovative approaches and a break from inherited practices.

In Chapter 3, a detailed literature survey will be given. Section 3.1 presents the most important work in the scheduling theory, while Sections 3.2, 3.3, 3.4 give an in-depth survey of widely used Grid schedulers, simulation tools and monitoring systems.

3.1. Scheduling Theory

3.1.1. Taxonomy of Scheduling

The general scheduling problem has been described in several seminal works [13, 14] and is a restatement of classical notions of job sequencing in the context of production management [15].The functionality of the scheduler can in broader view be described as that of a resource management resource [16]. It consists of a mechanism or a policy used to effectively and efficiently manage the access to a certain resource by a number of competing users.

Scheduler is a mediator between the resource and the consumer, and as such has to satisfy two opposing requirements: in term of quality of service the

performance expected by the user, and in terms of system utilisation the load fraction expected by the resource operator.

A useful four-category taxonomy of scheduling has been given by Flynn [17], and further discussed in [18]. Parts of the overall categorisation are shown in the diagram in Figure 2.

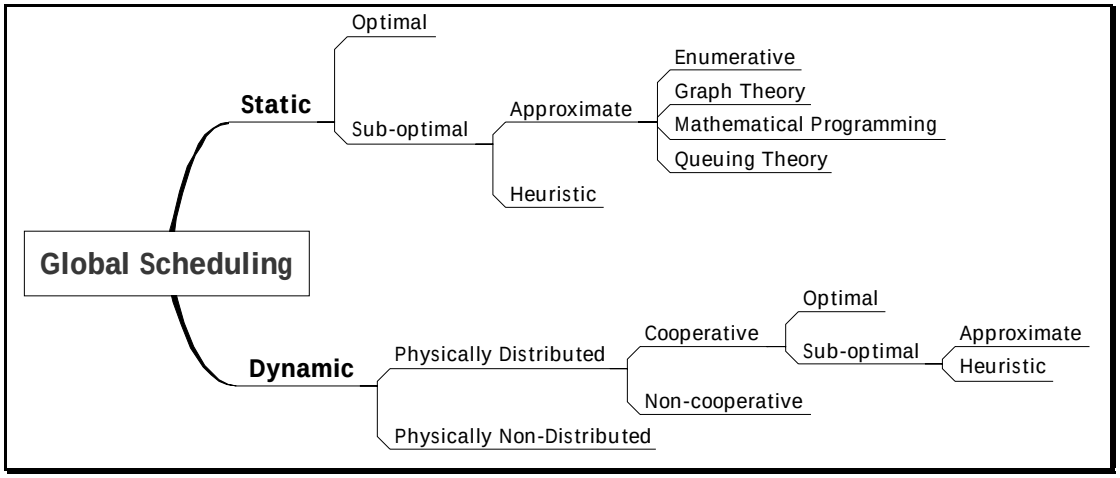


Figure 2 Taxonomy of Scheduling

However helpful this hierarchical approach is, it must be considered with caution, in relation to and from the perspective of a single point in the distributed computing environment. At the highest level, the taxonomy can be divided by scope into local and global schedulers. As I will discuss in later sections (3.2 Survey of Schedulers), Grid computing environment consists of many layered scheduling components and depending on the point of view taken same component can be seen as either a global or a local scheduler.

The scheduling research work proposed in this thesis would fall in the global, dynamic and physically distributed category. Depending on the nature of the Grid environment and the decisions that will be taken at the later stage of the research, the scheduler would use heuristic, or approximate methods, or a combination of both.

3.1.2. Predicting Application-Level Performance

Methods for predicting various aspects of application-level performance based on historic data have been previously researched in the context of high-performance parallel computing [19, 20]. Of particular interest were estimates of application queuing time, wait time (time between arriving at the head of the queue and starting execution), application run time and overall makespan (time elapsed from submission to the scheduler to the end of execution). Resource utilisation of the processes was studied in less detail, as HPC systems tend to be space-shared and jobs were usually allocated exclusive use of a part of the system.

Prediction methods used range widely depending on the scope and target use. Most widely used are [21, 22]:

- Last-value predictions

- Mean and Median based statistics
- Linear regressions
- Greedy algorithm
- Genetic algorithm
- Neural networks

Most commonly used algorithms are based on derivatives of mean and median based statistics. These fare well in relatively static pools of resources and well-behaved job distributions, but special care must be taken when handling multimodal and probabilistic environment such as the Grid.

Previous works on this topic have established factors which have important implication on the quality and usability of the forecasts [23]. In deducing patterns in the data set it was shown as important to build experience based on similar events, like-for-like data points. Jobs would need to be partitioned and categorised so that different predictions, and even different forecasting algorithms, can be used.

Timeliness of the predictions has shown to be at least as equally important as the margin of error [24]. As prediction algorithm complexity increases, the time taken for making a scheduling decision may significantly reduce the benefit in makespan time such prediction would create. If it all possible, the quality of the given prediction should be quantified, as this would increase the usability of the forecast.

To this date, little research is evident in the correlation between the job execution statistics, its related meta-data and the state of the computational environment at the time of execution. The lack of common accounting standards and log file formats further hinders comparisons of information obtained from various sources.

3.2. Survey of Schedulers

Despite the Globus Toolkit evolving as a standard Grid middleware, the meta-computing landscape is still very fragmented. The core functionality required from the middleware is not firmly defined, and different approaches lead to a blurred distinction between the scheduler and meta-computing middleware. Often, there is significant overlap: many of the surveyed schedulers can operate as either standalone systems, or as Globus job managers.

The following sections aim to present the most prominent and widely used schedulers in the Grid community. Various research projects are constantly developing new schedulers. These efforts sometimes result in little more then a conceptual implementation, but a few are developed into working implementations and are subsequently used in production Grid deployments.

3.2.1. AppLeS

Application Level Scheduling (or AppLeS) [25], was developed at the Computer Science and Engineering Department at the University of California San Diego (UCSD). AppLeS was one of the first Grid schedulers to investigate adaptive scheduling and provide full job support from resource discovery, schedule generation, selection and adaptation, to application execution. Its schedule generator can take into account and optimise for user's performance criteria, such as execution time, or target turnaround time. These benefits come at the price of having to extensively modify every application to be scheduled using AppLeS. In most cases this would require a joint team of scientist and AppLeS developers modifying and recompiling the source code to enable the application to be dynamically scheduled. The result is an integrated piece of software composed of a domain-specific component, a scheduling superstructure controlled by the AppLeS agent and an actual problem solving code. AppLeS developers have in this way enabled over a dozen applications, and this scheduling method is best suited to parameter sweep applications and master-slave divisible workload. Performance modelling methods are based on well known third party software (such as Network Weather Service see Section 3.4.3) and have to be manually customised to the application and hardware platform in question.

AppLeS can provide significant increase in resource utilisation and optimisation of requested performance metric on a properly tuned cluster. However, modification of the source code may not always be possible or desirable, and presents a high switching cost to the user. This will only be acceptable for high-value niché applications, or clusters based on expensive or exceptional hardware. To date, AppLeS developers have presented their work on application scheduling for synthetic aperture radar, parallel tomography and magneto-hydrodynamics, among others.

AppLeS bears significant differences to our approach in requiring each application, set of resources and prediction algorithm be adapted to its scheduling framework and domain in question. This requires significant effort on behalf of the user, cluster administrator, AppLeS developer and software provider. Any solution developed in such a way may not be portable, or may not perform sufficiently well even with minor changes in the cluster composition, network topology or usage patterns. Nevertheless, AppLeS has shown possible benefits of adaptive and predictive schedulers, and an obvious need for their development.

3.2.2. Condor–G

Condor[26] is a batch scheduling system targeted at harvesting unused computational cycles from a heterogeneous set of user workstations. It was first developed in 1988 at the Computer Science Department, University of Wisconsin-Madison. Condor suite[27] provides scheduling with different policies and prioritisation, resource monitoring and job management. Resource owners maintain full control of their hardware and can set policies on their

acceptable use. Condor uses a proprietary ClassAd language, which allows hardware owners to describe their resources, and users to specify arbitrary resource requests. A matchmaker component compares these to match job requirements with appropriate execution hardware.

Condor-G [27]is an extension to the core Condor components and functions as a bridge between a Condor pool and resources accessible through Globus middleware. Running as a Globus job manager on each Grid worker node, and as a controller on a dedicated node within the Grid, it translates between GRAM and ClassAd protocols, allowing jobs submitted to a Condor pool to be executed on machines in the Globus cluster. Due to its strong user base and developer community, Condor-G is widely accepted as a Grid job manager, and is deployed even when no Condor cluster exists.

Condor is a batch scheduler with first-in-first-out approach, although some job grouping and prioritisation is possible. Condor provides the ability for check-pointing and job migration, important aspects for operation on non-dedicated, commodity clusters. If a machine running a Condor submitted process becomes unavailable, whether due to the local policy or a malfunction, the job can be restarted from the last check point on another suitable worker node. Jobs submitted through Condor are sandboxed on execution nodes, and should local policies restrict file I/O, Condor can redirect low-level API calls to a remote file system.

Condor has a proven track record of providing high throughput computational clusters, and with its lightweight client deployment is of special benefit to institutions with large pools of underutilised workstations. However, it is a very much centralised system with clear separation between master and slave nodes. Its scheduling framework has no notion of deadline, and other than coarse-grained prioritisation, little intelligence can be added to the scheduling process. Despite additional improvements offered by Condor-G, the framework is not flexible enough to accommodate dynamic Grid resources distributed across administration boundaries.

3.2.3. N1 (Sun) Grid Engine

Sun has embraced the Grid as one of their core future technologies, and has developed its Grid software with a very strong business perspective. Sun's Grid platform is based on the Sun Grid Engine (SGE) [28], an open source software from which Sun has recently spawned a commercial product called N1 Grid Engine[29]. Grid Engine can function as a stand-alone system, or it can be used as a job manager within a Globus Toolkit Grid environment. Currently, SGE supports one of the widest sets of hardware and OS combinations: SPARC, AMD64, x86a and Mac hardware running most of the UNIX operating systems with even the support for Windows XP forthcoming.

Grid Engine is built on an agent based master-slave model. Master node serves as the only ingress point to the system and offers a command line and graphical user interface to the cluster. Slave nodes are running an agent responsible for

executing jobs, monitoring their progress, and communicating with the master. Each execution host in the cluster represents one queue and each CPU on that host is seen as one slot. The implication is that each CPU will only ever be assigned at most one job regardless of its utilisation. Creation of parallel environments is possible, with one job spanning across a number of CPUs and nodes. Queue scheduling is based on a policy and priority modified first-in-first-out model. Agents running on worker nodes report to the master details about underlying hardware, software and execution environment from which complex queries for resource selection can be made. Role-based user privileges can establish groups or projects with different priorities, execution schedules and billing options. Overall, SGE provides a stable production system for high throughput computing in commercial environments.

Sun Grid Engine approach is very centralised in nature, with a batch scheduling system at the core. It provides for a good degree of control over the use of resources, but no out-of-order or scheduling to a deadline is possible. The monitoring and management system lacks scalability, relying on approaches which would be inappropriate in large distributed systems (for example plain text accounting files needing manual rotation). Performance and robustness issues are raised by the use of a single master responsible for accepting and dispatching jobs. Experience shows that even a small worker node population (hundreds of workers) with a moderate job arrival times can saturate a 4-way SMP master.

University College London Central Computing Cluster (CCC) compromising 104 dual CPU compute nodes has deployed Sun Grid Engine version 5.3 as its Grid middleware. Preliminary user experiences have been positive with respect to stability and availability. The relatively low size and utilisation levels of the CCC did not expose any problems, or have significantly stressed the scheduler.

3.2.4. Nimrod/G

Nimrod [30]scheduler was developed as a tool for facilitating large runs of parameterised simulations over a distributed set of resources. It provides a declarative parametric modelling language used to create a plan for deployment of a large number of tasks. Nimrod application running on a master gateway is used to submit, execute and collect results from multiple worker nodes. Resources used are typically loosely coupled workstations with well defined usage periods (such as office hours), or dedicated resources on which a high level of utilisation should be maintained. Nimrod offers a Tcl based scripting language for job setup, and a simple graphical user interface through which ranges for parameter studies can be defined. The GUI also provides basic monitoring of running nodes and the progress of the overall experiment. A reengineered version of Nimrod, called Clustor, was commercialised by Active Tools until the company ceased to exist in 2003.

Nimrod scheduler was developed for a static set of resources, and could not be adequately used in a highly dynamic context such as the Grid. Nimrod/G [31]further developed the Nimrod concept, and embraced Globus middleware for dynamic resource discovery, job submission and security. Using Nimrod/G,

user can define a deadline for completion of submitted jobs, and/or a virtual budget available for computational resources. By offering a "budget" metric, Nimrod/G is looking to provide a framework for market based computational economy where such services could be traded.

Key components of Nimrod/G architecture [32]are its parametric engine, scheduler, dispatcher and job-wrapper. These components interface with Globus middleware, and provide resource discovery through MDS queries, job data staging through GASS and job dispatch through GRAM. Security services including authentication and authorisation are provided by GSI. During a schedule generation stage, Nimrod/G runs a sample of the parametric study application on the target nodes and uses measured hardware performance as a benchmark for later execution duration predictions and schedule generation.

Nimrod/G is one of the first schedulers developed specifically for the Grid environment. It was shown to offer good scheduling performance, with good adherence to requested deadlines. The trial run prediction method lends itself well to the heterogeneous nature of the Grid. However, Nimrod/G is aimed at parametric study applications, whose execution times are very narrowly distributed, and generally independent of the input parameters. By limiting its scope, Nimrod/G is able to utilise simple prediction methods to achieve satisfactory scheduling performance. Although these applications form an important group of scientific software presently running on the Grid, a general purpose scheduler must be able to handle other types of applications equally well.

3.2.5. Portable Batch System (PBS)

Portable Batch System (PBS) [33] is a widely used scheduler in large institutional clusters, and has become a *de facto* standard batch scheduling system in the Grid and Beowulf[34] environments. Originally developed to manage aerospace computing resources at the NASA, it now comes in two versions: an open-source implementation OpenPBS[35], and a commercial product from Altair Engineering PBS Pro[33]. PBS can function without any Grid middleware as standalone workload management system, or integrated with Globus Toolkit as a local scheduler. The system is available for almost any cluster system from vector and parallel supercomputers, SUN, SGI, HP, Alpha, and Macintosh workstations, to Intel and AMD based Windows systems.

Portable Batch System[36] is based on a centralised server-client model, in which server accepts job execution requests and forwards them to one or many clients for execution. The scheduling component is separated from the server process, and through the use of PBS API can be modified to implement different scheduling algorithms. The Scheduler communicates with the Server to obtain submitted job information, and with the PBS resource monitor to acquire utilisation data. It can operate on single or multiple queues and create schedules based on site policies, priorities and the utilisation state of the cluster. Subject to the underlying hardware and software support, PBS Pro can reserve resources in advance and schedule accordingly.

Although very robust and reliable, Portable Batch System is another implementation of a modified FIFO approach to job scheduling. It is best suited to well managed and controlled environment, with (mostly) homogeneous hardware and software, and with unified accounting and administration policies. PBS Pro has provisions that mitigate single points of failure (such as failover Server), and offers cross-system scheduling with access control lists and user mappings. However, it is clear these are seen as add-ons rather then core philosophy of PBS. Job recovery mechanism provided can restart jobs disrupted by a node failure on another host in the network, but the system is not designed to deal with a highly dynamic resource pool that a Grid environment may present. Most importantly, no deadline scheduling is supported, and job staging and execution times are considered unknown.

3.2.6. Load Sharing Facility (LSF)

Load Sharing Facility (LSF) [37] grew out of the PhD thesis of Songnian Zhou, who later successfully developed and marketed LSF as a core product of his start-up business – Platform Computing. The company has been at the centre of Grid development and standardisation efforts in the GGF, with an extensive user portfolio of blue chip companies in the banking, manufacturing and life science sectors.

Platform LSF is a commercial product, and little information on the inner functioning of the scheduler is given in the company's whitepapers. The author was unable to obtain a demonstrational copy of the product for testing purposes. Research papers mostly examine and compare LSF performance with other scheduling systems, and only very early works by Zhou [38]offer insight into the algorithms of the early versions. From the information available, the core of Load Sharing Facility is the Virtual Execution Machine™ which provides virtualisation of underlying resources and is the primary execution environment in the LSF enabled cluster. A web based, SOAP/XML enabled interface offers customisation and integration with other applications. According to the company web site, an element of self-management has been built into Platform LSF to offer guarantee zero downtime, self-adaptive dynamic allocation of resources, and self-healing to reduce management overhead.

Platform LSF offers a comprehensive set of scheduling policies with support for fair-share, pre-emptive and SLA based scheduling with advanced resource reservation. The implementation of these is not discussed in publicly available papers and LSF documentation. Information gathered from informal sources suggests good performance levels, and a considerable overall increase in resource utilisation and efficiency. Further research into LSF scheduling methods will be undertaken.

3.2.7. PACE/Titan

PACE/Titan [39] toolset was developed by the High Performance Systems Group at the University of Warwick, and is one few schedulers supporting out-

of-order scheduling composition for deadline execution based on performance predictions.

The PACE toolkit [19]uses pre-execution modelling and analysis techniques to predict the runtime and resource utilisation of an application on a given hardware platform. It relies on hardware and software characterisation templates, with an evaluation engine tasked with extrapolating expected application performance on the requested platform. The PACE toolkit requires all applications to be re-compiled and linked with PACE libraries, so that their performance templates can be created. Equally, each host with a different hardware, operating system or any performance-influencing component needs to be profiled before PACE can integrate it into its runtime predictions.

Titan [24]is a workload and workflow management component of the toolset. Using performance predictions supplied by PACE, Titan optimises the execution schedule to reduce idle time, makespan and scheduling delay, while maintaining deadline adherence. Titan uses a genetic algorithm with crossover and mutation to locate an optimal schedule. The algorithm is constantly run on the pool of outstanding jobs, replacing the current best schedule if a better one is found. Titan also provides management of inter-dependant tasks and jobs with sub-workflows, and is able to optimise their execution in order to minimise total runtime.

PACE/Titan toolset has been developed to the Open Grid Services Architecture (OGSA) standard, uses SOAP messaging and runs in a Globus Toolkit 3 container. Considering this open architecture, it should be possible to selectively replace parts of the toolset with third party components – for example a different prediction engine could be used to reply to Titan requests.

Several research papers by the developers of PACE/Titan toolset have presented good results of its predictive scheduling technique. Despite these, the main drawback of the toolset is the need to recompile applications, and to extensively profile target hardware platforms. For large number of users running various applications on non-dedicated resources, such as in a typical Grid scenario, this may prove very difficult or impossible. The main strength of the PACE/Titan scheduler is in running high-end scientific applications on relatively static pools of high performance dedicated hardware. For such clusters, the investment in deploying the system and adapting the applications is justified (for example Adaptive Grid Eulerian Hydrocode running on US Department of Energy Accelerated Strategic Computing Initiative resources). The reported performance of the PACE/Titan scheduler shows the value of runtime predictions and the need for deadline scheduling.

3.2.8. Imperial College e-Science Network Infrastructure

ICENI[40] has been developed by the London e-Science Centre at the Imperial College London as a generic and modular meta-scheduling framework able to use a variety of underlying Grid middleware.

The architecture [41]separates scheduling and launching frameworks, allowing each to be independently extended to support the widest array of deployment scenarios. ICENI aims to explore the role and the flow of meta-data in the computational Grids; a Performance Repository maintains data on the job execution times on different architectures and with different network bandwidths. ICENI prediction engine treats applications as a collection of simple components connected as a directed acyclic graph (DAG) with varying depths and dependencies. It introduces a user-defined benefit value, such as target execution time or computing cost, which the scheduling process aims to optimise.

The launching framework can be adapted to the middleware and hardware platform on which ICENI is running – currently supported systems are Globus Toolkit 2.4, Sun Grid Engine, Condor or simple *fork*. The scheduling framework supports multiple concurrent schedulers, and user selection of preferred scheduling algorithm. The schedulers assume no exclusive control over the resources, as these could be made available to other, local or remote users, through alternative access methods. An API is provided by the scheduling framework for retrieving the meta-data from the Performance Repository, and for functions common to all schedulers (such as performance predictions and resource discovery).

ICENI authors have developed four different schedulers for use with the ICENI framework. They have recognised that for jobs of shorter duration time taken to develop a schedule can be longer then the total execution time. For this class of jobs, a random scheduling algorithm is adopted, and an optimised version selecting best out of *n* random schedules is also provided. For more complex job sets, simulated annealing or Game theory schedulers are considered. In the comparative tests by ICENI authors, simulated annealing performed best, with the best of *n* random scheduling faring very well. Game theory did not produce good quality schedules, and was outperformed by random scheduling at the fraction of the schedule computing cost and time.

The ICENI project lays important foundations as one of the first schedulers developed specifically for the Grid, with its heterogeneous and space-shared resources, and dynamic resource availability. It parts from the traditional approach of the batch schedulers and offers out-of-order job execution. Although the importance of meta-data is considered, its integration in the overall flow of monitoring information could have been more thorough. ICENI falls short of offering fully fledged deadline scheduling, but optimisation of wall-clock job execution time can be done using the benefit function. The core ICENI scheduling work focuses on the development of a well performing scheduling algorithm, with little or no mention of the job execution time prediction methods, their accuracy and computational cost. Due to an open architecture and modular design, ICENI offers a good platform for deployment of third party components and their testing in a production-like environment.

3.2.9. Maui Cluster Scheduler

Maui Cluster Scheduler [42]is an open source scheduler primarily developed and supported by Cluster Resources Inc. Maui forms the basis of the company's other commercial offerings, Moab Cluster/Grid Suite and Silver, but its development was widely supported by the HPC community, U.S. Department of Energy, and many others. Maui supports a wide variety of cluster hardware, operating systems and scheduling APIs including PBS, LSF, Loadleveler, and Sun Grid Engine.

Maui Cluster Scheduler is a high level batch scheduling system, with support for scheduling policies, dynamic priorities, resource reservations and fair-share allocation[43]. It relies on lower level schedulers to act as resource managers and launching tools, supplementing them with additional monitoring and accounting functionality. Each job submitted is assigned to a queue according to the applicable policies and priorities. On job submission, the user is requested to state the maximum wall clock execution time, and if a job is part of a larger workflow define other prerequisites for that job's start. These parameters enable Maui to appropriately reserve required resources and construct an initial schedule. Further schedule optimisation are made using job prioritisation and fair share algorithms. Optionally, Maui can be configured to use backfill, an out-of-order scheduling strategy inserting shorter jobs into gaps created by mutually dependant large jobs. This method can significantly increase job throughputs and cluster utilisation, but may lead to users "playing" the scheduler and reducing its fairness.

Although Maui maintains accounting data on previous user-predicted and actual job execution times, this information is not used in any way. Its analysis reveals users are likely to grossly overstate the wall clock time required for executing their application padding their, initially poor, estimates to account for overloaded compute resource, prolonged data staging or unexpectedly complex computations. These effects compound to lower the accuracy of these predictions to about 30%. Backfilling algorithm, combined with hard resource limiting, leads to a decrease in scheduling efficiency for large compute jobs as any CPU time freed by jobs under-running is used for backfill. Overall, Maui is a stable and well supported scheduler for homogeneous compute environments, and offers significant increases in efficiency, manageability and fairness compared to similar batch schedulers. The lack of an autonomous and intelligent prediction system reveals the unreliability of user supplied predictions, and the importance of this data in creating effective schedules.

3.2.10. Others

Legion[44] was the first modern meta-computing systems, developed at the University of Virginia. If the Globus Toolkit can be considered as a "sum of services", the Legion toolkit offers a unified and integrated architecture. Legion did not achieve the wider popularity and community acceptance similar to Globus. It remains to be seen whether Globus' modular approach will provide a

robust and stable framework, or a more controlled and integrated system like Legion would be required.

NetSolve (GridSolve) [45] is a remote procedure call (RPC), agent based system for solving numerical problems. The system manages resource discovery and load balancing of a distributed set of resources. NetSolve has interfaces for C and Fortran, and commonly used scientific applications such as Matlab, Mathematica and Octave.

Ninf-G [46]is a reference implementation of the Global Grid Forum (GGF) recommended GridRPC specification. It is based on the Ninf system, and similarly to NetSolve aims to Grid-enable legacy application written in Fortran and C.

A suite of schedulers has been developed for a specific, niché, application. They offer good performance levels on specific installations, and running specific class of jobs. Some of these are:

- **Chameleon**[47] – improves scheduling in the Data Grid, and similar data intensive environments, by considering the amount of computational resources available, as well as the data availability.
- **NQE/NQS** – legacy batch schedulers, mostly run on mainframe machines. Being replaced with PBS or PBSPro (see Section 3.2.5).
- **MARS**[48] – a meta-scheduler for University of Michigan campus Grid.
- **Scheduling Expert Adviser (SEA)** – a 1997 project that converts a high-level description of a computational task supplied by a user into a set of facts and rules on which the scheduling is based.

3.2.11. Conclusions

Presented survey of schedulers clearly indicates that most of the scheduling approaches have been inherited from the legacy distributed clusters. These, almost exclusively batch systems, offer good levels of reliability and control while sacrificing utilisation levels, user experience and dynamic resource handling. Such trade-off is currently acceptable in the production environments mostly due to the lack of stable, usable alternatives.

Novel schedulers, specifically developed for the Grid, are emerging from several research projects (Nimrod/G, ICENI, Titan/PACE). Looking to tackle stochastic and probabilistic nature of Grid resources, these schedulers recognise the value and the necessity of forward looking estimates on the job execution times and its resource utilisation.

3.3. Survey of Grid Simulation Suites

3.3.1. SimGrid

SimGrid[49] has been developed at the University of Southern California, as a toolkit providing core functionality for simulation and analysis of scheduling methods for distributed and parallel applications.

The toolkit [50]provides several levels of abstractions, and has the ability to import topology specifications from third party applications. SimGrid is implemented as an agent based simulator, with each of the scheduling agents running at a certain location, communicating through a network path using a defined communication channel, and executing a given task. SimGrid abstractions implement these services as Agent, Location, Task, Path and Channel objects.

Simulation scenarios are executed in following steps: modelling the simulated applications by defining functionality of each agent, defining resources and allocating agents to appropriate locations, and running the simulation while observing different levels of trace verbosity. SimGrid supports compute and network low level resources, which can be of either fixed characteristics, or varying according to a trace file. This is particularly important in modelling network links, and a representative Grid topology can be simulated by using traces from network monitoring tools such as NWS. Compute resources are characterised by computational speed relative to a reference node and their availability (from 0 to 100%). Network links are described by their latency and bandwidth. These resources can be shared and contended for with three different strategies: first in first out (FIFO), first ready first out (FRFO), and shared (user implemented fair share method). Once the simulation scenario and hardware topology has been developed, different scheduling techniques can easily be implemented and repeatable measurements made to assess their merits.

SimGrid builds on best approaches from more complex and specific simulators, while maintaining simplicity and good performance levels. Its use by a number of research projects, and numerous publications of SimGrid simulated results have confirmed it to be scalable, configurable and extensible enough to simulate a wide variety of scheduling problems[51]. Validation of SimGrid results remains a difficult question, especially in a relatively new setting that the Grid is. The problem is alleviated to some extent by the fact that SimGrid is based on models previously accepted in the scheduling community.

3.3.2. MicroGrid

MicroGrid[52] is a simulation tool developed at the University of California, Sand Diego, under the sponsorship of National Science Foundation as part of the Grid Application Development Software project (GrADS) [53]. MicroGrid is an online simulator, providing a virtual Grid environment on which real Grid middleware and Grid applications could be run. Currently Globus Toolkit is the

only supported Grid middleware, but the latest version of MicroGrid is able to run Grid applications written in MPI, C, C++, Python and Perl.

Primary concern with an online simulator such as the MicroGrid is the implementation of virtualisation functions. MicroGrid relies on the operating system to provide unique namespaces and seamless sharing, and only virtualises the host identity[54]. This is achieved by mapping virtual IP addresses to host's physical address, and trapping all resource related calls to perform the translation.

Physical resources in the simulated virtual Grid are discovered and characterised through an extended and virtualised Globus Information Service (GIS). Computational resources are defined using a scaling factor to their real performance, denominating the slice of the CPU time that will be used in the virtual Grid. Network elements are simulated through an external application (VINT/NSE), creating high overheads and limiting scalability.

MicroGrid simulator has been validated by the authors, in different testing scenarios ranging from single component tests to running a fully fledged Grid application (Cactus Application) [55]. The virtual Grid approach is helpful in situations where application behaviour is hard to model, or when unfeasible test scenarios are needed – such as investigation of catastrophic node or network failures. MicroGrid requires stable middleware and application suites, and as such could not be used in early stages of a novel scheduler development. The need for global coordination of resources in the virtual Grid enforces a "maximum feasible simulation rate" on the whole environment, dictated by the lowest specified physical hardware on which MicroGrid is running. Although theoretically possible, large Grid simulations with complex resource pools could be prohibitively time consuming to execute.

3.3.3. GridSim

GridSim [56]is a simulation tool developed at the Monash University, Australia and freely distributed under the open source license. GridSim is based on SimJava2[57], a process based discrete event simulation tool for Java. Developed by the same team as the Nimrod-G scheduler means that economy-based scheduling and resource allocation principles have been deeply embedded in the rationale of this simulation package. GridSim focuses specifically on modelling time- and space-shared resources, with support for concurrent tasks running on the same resource[58]. The geographical and social aspects of the Grid environment can also be modelled through variable resource background utilisation based on time zone, busy hours, days of the week and calendaring.

Simulation set up steps in GridSim include creation of resources, definition of applications (called Gridlets), and coding of resource brokers and the scheduling algorithm. Compute resources are defined as processing elements (PE), basic building blocks whose performance is specified in MIPS and can be coupled to form SMP-like architecture. Characterisation of network links was poorly documented, with only a reference to their "data transfer baud rate". Job

specification is done through Gridlet objects, which include explicitly defined computational cost (in MIPS), size of input and output data sets, preferred scheduling policy and user's deadline and budget constrains.

Although based on an already established simulation platform, GridSim is not as methodological in simulating realistic network topologies, link congestion, resource contention, and parallel applications as SimGrid. Poor documentation further mars development of genuinely useful simulations. Despite being a general purpose Grid simulator, GridSim is targeted at parametric research applications and economy driven scheduling approach. The authors have developed a GridSim based simulation of their Nimrod-G scheduler (see section 3.2.4), but few other projects have reported on their experimental use of GridSim, or on validation of simulated results.

3.3.4. Conclusions

Few Grid simulation tools are presently available, leading to significant problems in testing new scheduling approaches. All three simulators surveyed adapt a different approach, and are targeted at simulating different aspects of the Grid middleware. Generally, simulation of application behaviour is poorly captured with few realistic models, and little support for modelling of job statistical properties. Effective testing of new, probabilistic, schedulers will depend on being able to accurately simulate real application behaviour with its important statistical properties.

3.4. Survey of Monitoring Systems

3.4.1. Ganglia

Ganglia Cluster Monitoring[59] was developed at the University of California Berkeley, and was sponsored trough National Science Foundation's NPACI program, before becoming part of the PlanetLab project.

Ganglia aims to consolidate monitoring information in a hierarchical structure, and presents increasingly detailed data going from a federation of clusters, or Grids, down to a single worker node. It leverages widely used technologies such as extensible mark-up language (XML) for data representation, external data representation (XDR) for portable data transport, and round-robin databases (RRDtool [60]) for data storage and visualization. Ganglia has been ported to an extensive set of operating systems and hardware architectures. Throughout extensive deployment in production clusters containing over two thousand nodes [61], Ganglia was proven as a stable, robust and scalable system with low overheads.

Metrics monitored by Ganglia vary depending on the operating system and hardware support, but a core set includes processor load, memory usage and network performance. Due to the modular code design, Ganglia is highly

customisable and can be modified to monitor custom metrics specific to the local environment. By either modifying the source code, or by using Ganglia's custom metric publishing tool, these can be integrated in the core metrics' information flow.

Cluster nodes running Ganglia can either publish their measurement data, collect data published by other nodes, or do both thus creating a distributed data repository. Low overhead communication is implemented through broadcast messages within the cluster, or unicast links between the clusters. Data storage is handled by fixed-size round robin databases, and a Perl toolkit is provided for data visualisation through a web based interface.

Despite being efficient way of storing historical data, round robin databases loose detail and alter statistical properties by consolidating older measurements using simple functions such as averaging or min-max. Analysis of resource utilisation, process behaviour, and proposed scheduling algorithm would depend on statistical properties of historical data for correct operation. If Ganglia collected data is to be used for process resource utilisation trending and pattern matching, methods for preserving highest detail data would need to be developed.

3.4.2. Relational Grid Monitoring Architecture

Grid Monitoring Architecture (GMA) was developed by the similarly named working group under the auspices of Global Grid Forum, a worldwide forum for Grid developers and users. The working group was focused on producing a high-level architecture statement of the components and interfaces needed to promote interoperability between heterogeneous monitoring systems on the Grid. Relational GMA (R-GMA) [62] was developed as a web service implementation of the GMA specification, providing access to monitoring information through a relational database concept. After initial development as part of the European DataGrid project, R-GMA is now a candidate system for use in the "Enabling Grids for E-science in Europe" project [7].

The GMA working group has recognised that performance monitoring information differs from other forms of system or program-produced data: it has a short lifetime, is frequently updated and is stochastic in nature [63]. The group's subsequent recommendations proposed a monitoring architecture consisting of three components. Data Producers would publish their capabilities in the Directory, and provide information directly to the data Consumers based on their subscription to particular information feeds. Such approach implies a separation of the meta-data describing the monitored metric and the stream of actual measurement data. Relational GMA system builds on this model by relieving consumers and producers of registry interaction, and by providing a relational database communication model between the two. However, R-GMA is not a general distributed relational database management system (RDBMS); it rather provides a method of applying a relational data model in a Grid monitoring environment.

R-GMA is based on industry standard SQL database, and imposes a standard query language and database schema. It can thus benefit from proven scalability and robustness of these components. Information flow and component interaction is based on SQL CREATE TABLE, INSERT and SELECT queries on virtual tables maintained by the Registry.

Grid Monitoring Architecture specification provides a bare framework for which adequate information providers and consumers need to be developed. Although the whole Grid community would benefit from its wider adoption, few installations use it. The EGEE project, R-GMA's biggest proponent, and its monitoring database may contain significant amount of data which could be of great use in understanding Grid applications and their execution time patterns. Although the usage of SQL databases mitigates reliability issues, the Registry and the database schema could be a single point of failure, unless properly replicated.

3.4.3. Network Weather Service

Network Weather Service (NWS) [64] originated at the University of California, Santa Barbara, as a monitoring and forecasting system for meta-computing environments. Since its first versions was published in 1997, it has seen many improvements and modifications to become one of the most widely used tools in the distributed computing area. NWS operates a distributed set of sensors and supporting processes, monitoring network resources, and collecting historical performance data. When requested, it uses numerical methods to generate forecasts for some future time frame based on previous monitoring data[65]. The aim of NWS was to enable better scheduling in meta-computing environments by predicting the real level of performance at the application level. Although NWS was developed primarily as a network latency and bandwidth monitoring tool, its open interface allows for addition of third party sensors. Discussion of NWS herein, and its features, is based on the version 2.8.

Network Weather Service system architecture[66, 67] is based on four separate components: Sensor, Forecaster, Name Server and Persistent Storage. Of critical interest are the Name Server process, which runs on one machine only and provides a directory capability, and the Persistent Storage process which stores and retrieves measurements. Name Server is the only well-known address used by the system, allowing for both data and services to be distributed, but also creating a single point of failure. NWS developers plan to migrate the Name Server to a distributed LDAP-based service and eliminate this problem. Data storage is implemented using circular data files, and no measurements are kept indefinitely.

The issues of measurement intrusiveness and reliability were often raised in the NWS user community, and newer versions have gone to some length in fixing them. CPU utilisation sensor now uses both passive (based on UNIX *vmstats*) and active monitoring (by running a compute-intensive probe). The sensor has an adaptive, heuristic algorithm tracking the discrepancy of passive and active monitors to decide on the right balance of the two. Intrusiveness is especially important in network monitoring; NWS network sensor has developed advanced

techniques for measuring end-to-end bandwidth and latency while maintaining a minimal impact. Network sensors organise in hierarchical *cliques* and perform mesh measurements within these, and point to point measurements between different cliques and hierarchical levels. Co-ordination of measurements is performed by a token passing protocol using adaptive time-out discovery, and with algorithms in place to deal with token loss due to network segmentation or node failure.

Prediction algorithms used in the Network Weather Service fall in three basic categories: mean based, median based and autoregressive methods[65]. Additional Forecaster modules can be developed to increase the quality of predictions, for predicting specific sensor metrics, or for operation in special environments. NWS operates a competitive environment for different Forecaster modules, requesting each to produce their prediction every time a forecast is required. Prediction errors of each module are tracked, and the one with the lowest cumulative error is selected for further predictions. In this way, NWS automatically identifies the best forecasting technique for any given resource[68].

Network Weather Service is targeted at predicting wide area network performance, and although it can be extended with custom sensors it does not render itself immediately to process execution time predictions. Circular storage methods used are similar to round-robin databases used by the Ganglia Cluster Monitoring, but provide even less historical information. Self-selection of the best prediction method is a novel and useful feature, but it also impose significant computational overhead on prediction calculation. This may not be relevant with simple mean and median based predictors, but may become so if a more complex Forecaster is developed. Regardless, NWS presents a seminal work in monitoring and resource utilisation fields, and clearly demonstrates the value of insight into historical monitoring data. Its simple, yet effective, prediction methods show that even moderately accurate predictions can be used with great success.

3.4.4. Other

Several other monitoring systems are used in the Grid community, usually with a more specific focus on one of the aspects of the system's operation. Often, large projects assemble toolkits of loosely coupled best-of-breed components, and distribute them as part of their customised Grid middleware.

GridMon[69] is a UK e-Science project monitoring network performance between each of the regional e-Science nodes. Using a suite of tests based on simple *ping* scripts and Iperf utility, GridMon confirms connectivity and measures packet loss, round trip time and TCP/UDP throughout. A more intrusive test establishes end-to-end application level performance by copying large files (1 to 80 MB) using SSH file transfers. All measurements are done from each node to each other node, thus creating a mesh matrix. This approach leads to a very intrusive and non-scalable network monitoring, appropriate only for a current small number of e-Science centres (12-15). GridMon publishes its measurements using a Web based visualisation suite, LDAP service or OGSA compliant web service.

Condor Hawkeye[70] is an extension of the Condor high-throughput cycle harvesting system (see Section 3.2.2), and is based on the ClassAd messaging protocol used in Condor-G. Hawkeye configures Condor pool master to periodically run a selection of scripts which take measurements and generate appropriate ClassAd messages. These monitoring messages can then be used to execute complex selections or conditional queries when submitting jobs. Hawkeye integrates well with a large installed base of Condor pools, and requires little administration effort. However, due to the (in)frequency of measurements, it is more of a summary utilisation and problem reporting tool than a high resolution resource utilisation monitor.

3.4.5. Conclusions

As with other management components, Grid infrastructure has inherited monitoring applications from the cluster and main-frame world. Currently, most monitoring systems will deploy the probes statically, perform the measurements in predefined frequency, and use static data retention policies. All data collected is treated and retained equally, and no differentiation is made based on temporal or relevance criteria. This framework creates a rigid structure in which no adaptation to the granularity, frequency, or communication parameters is possible based on the operating environment conditions. Apart from introducing overheads[71], this approach can not monitor a large scale distributed and dynamic environment effectively, and would not be able to concisely present required information when and where it is most necessary.

4. CHAPTER FOUR

SO-GRM PROJECT

The author's research work is part of Self-Organising Grid Resource Management (SO-GRM) project[72], sponsored by Engineering and Physics Research Council (EPSRC) and in collaboration with BT Research labs. SO-GRM is base research project aimed at developing an autonomous management infrastructure able to support the job execution through its full lifecycle – from job admission through scheduling and resource discovery to security monitoring. Every component of the SO-GRM architecture shares the same objectives: removing single points of failure through a distributed approach, reducing the administration load by using policy based management and creating agile, on-demand system through the use of self-organising principles.

The SO-GRM aims to present a platform for integrated testing of scheduling, simulation and monitoring components developed by the author. A Grid testbed has been deployed by the author (see Figure 3), and runs a full set of Globus middleware, supporting applications and SO-GRM components. Although of a limited size, the testbed should reflect a real Grid production-like environment and offer a good opportunity for *in-situ* testing.

This chapter briefly introduces the components so far developed by the contributors of the project (including the author), and reports on the functionality tests undertaken on the project's Grid testbed. Section 4.1 discusses the problem of Service Level Agreement management within the Grid; Section 4.2 presents a novel resource discovery protocol based on small-worlds topology; while Section 4.3 describes a novel distributed intrusion detection system. In Section 4.4 test set-up, methodology and results are discussed, while Section 4.5 concludes this chapter.

4.1. SLA Management

As the highest level management component in the system, Service Level Agreement (SLA) Management (SLAM) [73] is responsible for negotiation of service level agreements between commercial Grid operators and users, and among commercial Grid operators themselves. In the context of our research, an SLA describes the expected performance of the system from consumer/user's point of view. As we seek to provide an intuitive interface to the system, an SLA might not be expressed in explicit terms, stating for example the number and specification of required machines, minimum network bandwidth and latency required or similar. It is more likely that a user-level metric such as percentage of successful web hits, or deadline for application execution will be used. SLAM is expected to provide methods for translation of those abstract service level objectives into implicit low level resource requirements. Once these are obtained, admission control is undertaken to check the subscribed load of the cluster and assess whether such an SLA can be honoured. If that is the case, it is assigned a unique SLA identifier, used to tag each subsequent job relating to that SLA. Current version of SLAM provides a Type of Service (ToS) indicator that can be used by operators to prioritise certain jobs or ensure certain resources are reserved for higher value SLAs. On acceptance of a new SLA, a new Virtual Organisation (VO) is populated and serves as a container for all resources assigned to that SLA. This VO is populated through resource discovery procedure, and can dynamically grow and shrink within the bounds set by the service level agreement. This adaptive behaviour enables better utilisation through controlled oversubscription of resources.

SLAM is implemented as a Java application and runs on a single node in the network. Basic operation has been confirmed in the live test runs, and new functionality and improvements are being added.

4.2. Resource Discovery

Self-Organising Resource Discovery (SORD) [74] component is tasked with the discovery of computational resources which satisfy conditions set by SLA management and the scheduler. SORD is a distributed protocol in which each node acts autonomously to discover the most suitable host to serve a specific request. Initially each node is connected to a number of topologically near nodes (called neighbours) and few random far nodes thus creating a small-world topology [75]. These topologies have previously been considered in the problem of routing with local information and allow distribution of information to the correct recipient by using shortcuts. The protocol uses XML encoded query-reply and advertisement messages with limited time-to-live, and stores received responses in fixed size cache tables. By using different tables for each of the queried metrics different virtual topologies are created, and certain nodes evolve as most frequent successful candidates for fulfilling requests for such metrics (i.e.

node with much more physical memory than other nodes in the network will be a firm favourite for memory intensive applications). Main objectives in the design of the protocol where scalability and resilience to single node failures, both of which have been successfully met. The protocol implementation and integration with other components and Globus middleware was tested on the SO-GRM testbed, while extensive simulation has been done to examine its scaling properties. More information on scalability and successful resource discovery rates can be found in previous publications by Ioannis Liabotis [76, 77].

4.3. Integrity Information Intelligence - I^3

Integrity Information Intelligence (I^3) is a distributed run-time intrusion detection system. I^3 is a combination of anomaly and misuse detection systems: initially trained with the features of a well behaving process, I^3 is subsequently able to recognise suspicious utilisation patterns. Suspect patterns are classified as either re-occurring offending bad behaviour, or a new ambiguous feature. In the later case, one-off human intervention and classification is required. The feature set defining an anomaly is stored locally, with all other nodes in the network immunised by broadcasting the anomaly's definition as an XML antidote. Raw monitoring data is processed through the feature extraction algorithms that calculate central moments, and based on these mean, standard deviation, skewness and kurtosis. Outputs of these functions feed the classifier component which treats them as points in a multidimensional space. The classifier uses mahalanobis distance [78] to compare the level of matching between the training set and obtained data, and hence decide on the nature of the observed pattern.

I^3 agent is implemented as a Java application running on each node in our test Grid. In both simulation and testbed deployment I^3 has provided process classification with less than 1% error rate for a suitably configured threshold detection value. More information can be found in [74, 77, 79].

4.4. Functional and Integration Testing

Further to previous simulations and isolated testing of SO-GRM components, a functional and integration test of the overall system was performed on the Grid testbed. SORD, I^3 and SLAM were developed in Java, exclusively use XML for message passing and data storage, and have been previously tested independently to ensure proper core functionality. It was therefore decided to use eXist[80] XML database as data storage on local node and global VO levels. This was primarily a choice of design convenience and deployment speed; SO-GRM management system can be adapted for used with any other data storage method. Architecture block diagram is shown in Figure 3.

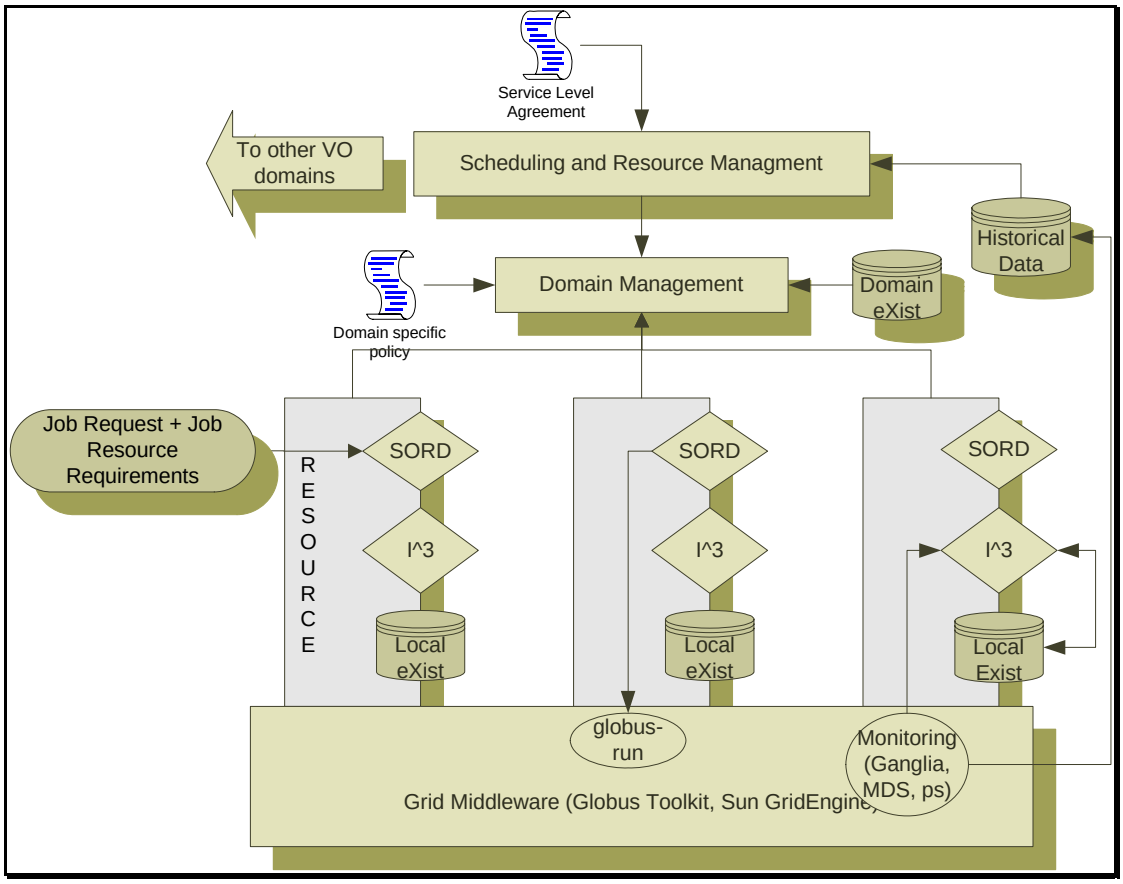


Figure 3 SO-GRM Architecture Block Diagram

Each node ran a full set of components with multiple management nodes maintaining domain and VO-level SLAs, policies and accounting data. The integrated monitoring suite, developed by the author and further described in section 6, provided all the measurements data. Grid application load was simulated using a suitably parameterised GridLoader tool, also developed by the author and detailed in section 5. Resource utilisation measurements and other monitoring data was collected and parsed into eXist XML database for consumption by all other components.

SO-GRM demo was run on a test bed consisting of six machines in the Department of Electronic and Electrical Engineering at UCL and six machines at the BT Research Lab at Adastral Park. Machine specification and installed OS and software components are given in Table 1.

Specification:	UCL Domain	BT Domain
CPU	AMD Athlon 2400+	PIII 550
Memory	512MB	256MB
Network	100Base-T Switched	100Base-T Switched
OS	Red Hat 9	Red Hat 8

Table 1 Demo Testbed Specification

The two sites were connected through a routed ATM link which was only partially under our group's control. Together with other administrative issues this ensured the demo was performed in a production-style environment, with sites residing in two distinct administrative domains and on two separate networks. Testbed network diagram is given in Figure 4.

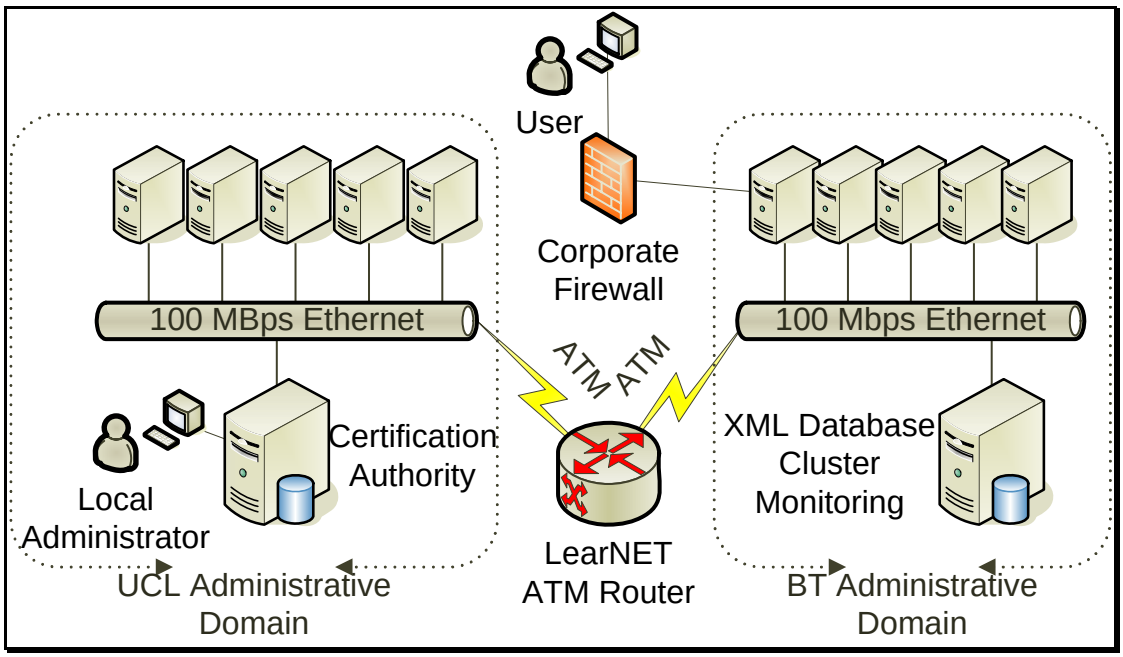


Figure 4 SO-GRM Testbed Network Diagram

The main purpose of the demo was to confirm end-to-end functionality and verify component integration. The test scenario called for establishment of a new SLA to support jobs arriving at the nodes in the BT domain with ToS requiring dedicated use of the machines. UCL was to supply secondary resource pool for jobs overflowing from BT's domain, or jobs whose resource requirement could not be satisfied within BT's domain (typically high physical memory requirements). As all components run in full debug mode, performance issues were of secondary interest. Figure 5 shows the output of SORD debug windows in process of querying the neighbouring nodes for resources according to the submitted request. In the top left of the screen CPU monitors show around 70% loading on three out of four hosts; remaining unloaded node will be used for the next incoming job.

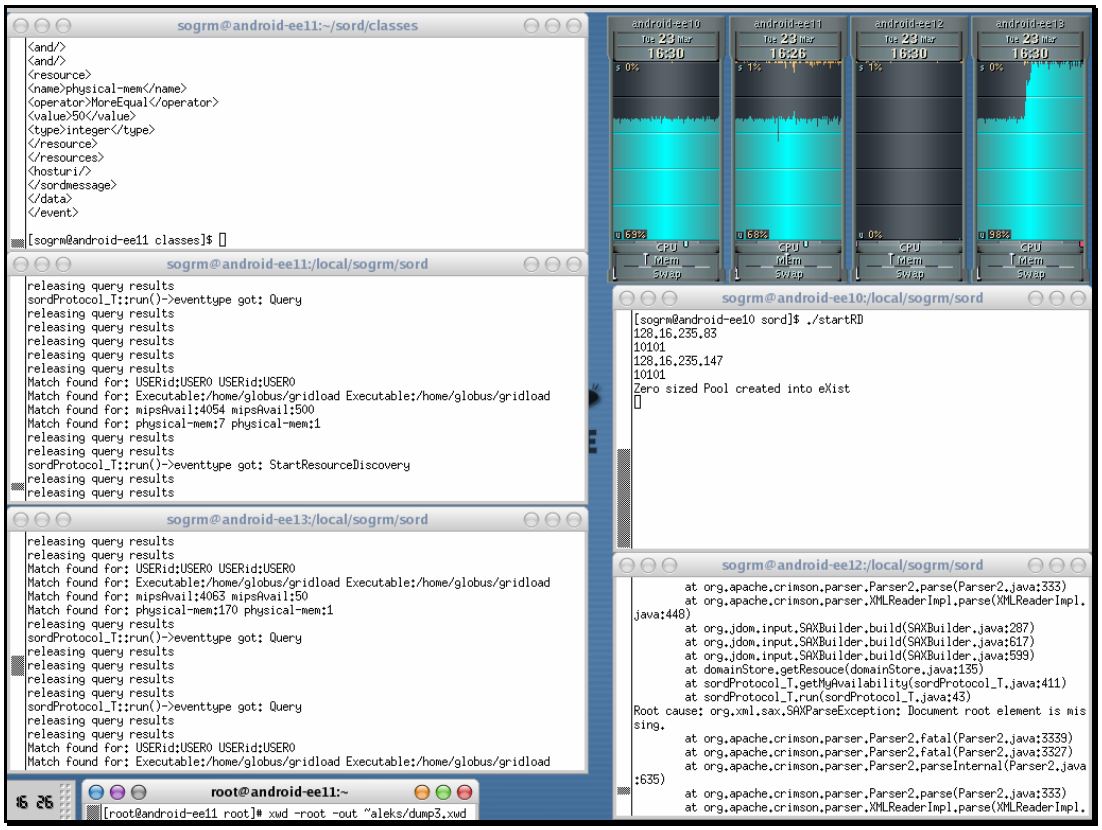


Figure 5 SO-GRM Demo Screenshot

First demonstration was run in December 2003 with subsequent runs and tests taking place until February 2004. Each component was scrutinised and further work needed was identified, as shown in Table 2.

SORD	
✓	Able to receive and interpret XML resource discovery requests
✓	Able to communicate with other SORD agents
✓	Able to query XML database for resource utilisation measurements
✓	Able to send jobs for execution to discovered target nodes
×	Resource pool too small to produce self-organising topology
I3	
✓	Able to obtain high-frequency CPU and memory utilisation data
✓	Able to store formatted data into XML database
✓	Able to monitor processes and perform anomaly detection
✓	Able to communicate with other I3 agents and exchange antidotes
SLAM	
✓	Confirmed basic functionality, proof of concept
✓	Able to acquire cluster overview from Ganglia XML repository
✓	Able to accept SLA negotiation request
✓	Able to mark up job requests and nodes with appropriate SLA id
×	Negotiation and prediction engine in development
×	Historical performance repository not yet implemented
Measurements and Monitoring (Author's Contribution)	
✓	Confirmed functionality of Ganglia Cluster Monitoring

✓	Confirmed functionality of external information providers
✓	Able to provide responses to cluster state queries
✓	Able to retrieve and customise the storage of high-frequency
GridLoader (Author's Contribution)	
✓	Confirmed proper operation on all target platforms
✓	Confirmed proper block and network I/O, memory allocation
×	Requires more testing of adherence to requested load parameters

Table 2 Conclusions of SO-GRM Test Runs

Overall, demonstration was successful in proving correct integration of components and served as a proof-of-concept for the overall management structure. The testbed continues to be maintained by the author, and will be used for further testing of GridLoader, the measurement and monitoring application suite, and the scheduling framework.

4.5. Conclusions

Self-Organising Grid Resource Management (SO-GRM) project has been presented in this chapter as a platform for integrated research in Grid resource discovery, scheduling, and security. The project components rely on autonomous, self-organising and distributed concepts to deliver a scalable Grid resource architecture with high degrees of self-management. The components have been deployed on the testbed, their functionality was confirmed and their performance tested. Areas requiring improvement, and the direction of possible further work, have been identified.

5. CHAPTER FIVE

GRID APPLICATION SIMULATOR

The following sections present the author's work on the Grid application simulator, called GridLoader. Section 5.1 briefly reiterates the motivation for development of such a tool (already presented in 2.3), section 5.2 captures the requirements, and section 5.3 presents the implementation of the GridLoader. Results of functional and qualitative testing are given in section 5.4, while section 5.5 gives direction for future work and concludes the chapter.

5.1. Motivation

The simulation tools available in the Grid research community, as surveyed in Section 3.3, could not fully satisfy the requirements for testing and optimising author's proposed probabilistic scheduler. GridLoader application was motivated by the need for a controllable and tuneable load generator, able to simulate the job statistics of applications run in the production Grid environments. Such tool would allow testing of the scheduling algorithm, monitoring component, and all aspects of the SO-GRM management framework in a realistic usage scenarios, without the problems usually associated with running on a live production Grid system.

Through author's work on a suite of monitoring applications, and its deployment on a production Grids running various application, real-life statistics on job arrival rates, their duration, distribution and resource utilisation will be obtained. With this information, it will be possible to parameterise GridLoader to present a realistic load to our scheduler, and compare its performance against the scheduler used on the production system.

5.2. Requirements

To represent a realistic Grid application, the GridLoader was required to simulate processor utilisation, memory allocation and network utilisation. The execution of the GridLoader would have to be fully parameterised, with suitable tools to facilitate orchestrating large simulation runs. Such deployment tool would decouple the overall statistical properties of jobs submitted to a cluster (whose CPU utilisation could be modelled as a skewed exponential function for example) from the resource utilisation statistics of a single node (on which an application's CPU utilisation could be modelled as a normal distribution).

One of the approaches for simulating a realistic application load, often used by benchmarking applications such as SPECmark, is executing a representative set of application code snippets in an automated way. This method gives a degree of repeatability[81], enabling comparison of hardware implementations by maintaining an unchanging application load.

The probabilistic and self-organising nature of the SO-GRM components would require a more fluctuating and dynamic testing environment.

Although similar is possible using trace-replay simulation systems (see SimGrid section 3.3.1), author's aim is for the GridLoader to be able to create a statistical distribution of similar loads, maintaining a level of ambiguity and challenging self-organising and adaptive components.

An important requirement was achieving the right balance between deterministic and probabilistic modes of operation. The simulation runs should be repeatable, and all simulation parameters should be adhered to if any incremental improvements to the management components are to be recognised. At the same time, a probabilistic element in the simulated application behaviour is required for a realistic and diverse environment to form, and for components' adaptability and self-organisation to be exercised. Different resource may also have to be simulated with different distribution functions and parameters – network transfers may have a substantially different statistics than the CPU utilisation.

The GridLoader would need to be submitted through Grid middleware on the target site just like any other Grid application. To reduce administrative and portability issues, a simple and portable code running under user privileges would be highly desirable.

5.3. Implementation

GridLoader is based on a state machine, with different states representing CPU, memory and network loading stages. UML diagram showing this structure is given in Figure 6. Current version of GridLoader uses a deterministic state transition table, progressing through network loading, memory allocation and CPU utilisation states in progression. This is similar to an "embarrassingly

parallel" [82] Grid application, such as a parameter sweep tool, staging the input data, allocating required memory and executing a CPU intensive core calculations that would usually produce a small result data set. A more sophisticated model is possible when GridLoader is used with probabilistic state transition table where all three primary states are entered into many times with changing probabilities. Although this behaviour is more realistic, and representative of a more complex Grid application, it creates a very dynamic environment for all other components and possible faults are hard to locate and debug. This mode will be used in advanced stages of scheduler operation testing.

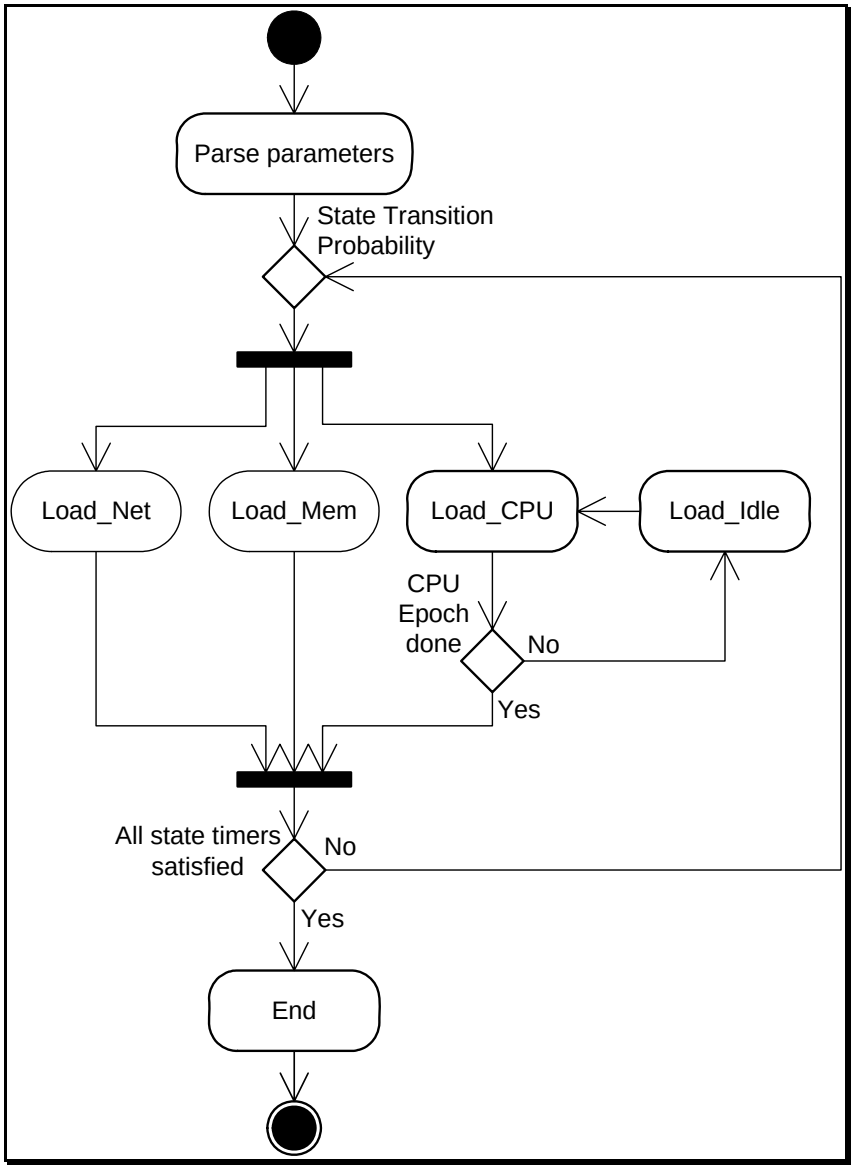


Figure 6 GridLoader UML Diagram

5.3.1. Application Simulation Stages

The network loading stage is entered into first. A message 1400 bytes long is generated, the required value for the duration of the network transfer is stored in the real-time countdown timer, and a UDP socket is opened to the specified IP

address. A loop is then entered in which the same packet is resent over the network with an intra-packet delay as parameterised at run time. On timer signalling the required time has passed, the socket is closed and a flag set for state transition.

Memory allocation state requests the kernel to increase the memory allocation to the process by the amount specified at run-time by using *malloc* function call. UNIX memory management is handled very differently depending on the implementation and kernel optimisation options, and may prevent a user process from directly managing memory allocations. GridLoader ensures that physical memory is actually allocated to the process by writing random data into the virtual memory space allocated by the kernel. The memory is freed during final clean-up state of the application, once all loading targets have been met.

Computational intensive part of each Grid application is simulated in the CPU loading state. This state contains two real-time nested timers, one keeping track of the total amount of wall time spent in the CPU loading state, and one tracking short time slices in which CPU is toggled between full throttle utilisation and idle. Very frequent swaps between these two stages result in a smoothed fluctuation of CPU utilisation when observed with above 100ms sampling period. Wall time duration is specified at run-time, while the duration of each run-sleep cycle is determined in a random manner using a Pareto (or other) probability distribution function. This function is randomly seeded at runtime, and partly parameterised through a command line option. Benefit of this approach is that even for equally parameterised runs, actual CPU load trace can be significantly different. This was an essential requirement for the testing of I^3 (see Section 4.3) feature extraction engine: GridLoader was able to simulate anomalies in process behaviour and test I^3 malicious process detection rate.

Once all timers indicate that requested loading metrics have been met, final clean-up stage is entered in which the allocated memory is freed, network sockets closed, and a log file with details of the execution written. GridLoader can also operate in a debug mode which produces detailed information about the state machine and execution timers.

5.3.2. Parameterisation Options

GridLoader was written in ANSI C, does not use any low level function calls or custom libraries. It compiles successfully on Solaris and Linux platforms. All parameters of the GridLoader's simulation can be supplied either on the command line, or from a configuration file. Screenshot in **Error! Not a valid bookmark self-reference.** shows the help screen with the required command line format.

```
[aleks@android-ee11 gridloader]$ ./gridload
Grid Loading Application, v.0.9.

ERROR: Must have exactly 5 runtime parameters.

Gridload utilisation:
    gridload <NET> <CPU> <MEM> <BURST> <IP> <PARETO_B>

Where:
    <NET> = NET transfer time in seconds - FLOATING POINT
    <CPU> = Total CPU loading time in seconds - FLOATING POINT
    <MEM> = Amount of memory to allocate in MB - INTREGER
    <BURST> = Interpacket sleep time in useconds - INTREGER
    <IP> = IP address to send network traffic to - DOTTED NUMERICAL
```

Figure 7 GridLoader Screenshot

The run-time parameters have the following meaning:

- NET – Total time for network transfer state, expressed in seconds
- CPU – Total time of CPU loading state, expressed in seconds
- MEM – Integer MBytes value of total physical memory to allocate
- BURST – Inter-packet delay time, expressed in μseconds and used to control the amount of bandwidth used by the network transfer state
- IP – Numerical IP address of the peer (or sink) for the network transfer state
- PARETO_B – Pareto parameter B used to influence the idle time transitions in the CPU loading state. Large value of this parameter cause the long tail of the Pareto probability distribution to extend, leading to spikier CPU utilisation trace and larger average levels of CPU utilisation. Subsequent runs with the same value of parameter B will not produce equal traces due to different seeding values of the random number generator.

To give overall cluster loading a certain statistical property, and to facilitate generation of configuration files for larger GridLoader runs, an auxiliary application was developed in Matlab™. Two types of parameters can be defined with either global or local scope. Global parameters influence the overall behaviour of the whole set of GridLoader jobs in a specific simulation run. These are used to coordinate the job set, and include the following:

- CPU_TOTAL_PARETO_[A/B] – Defines the value of Pareto probability parameters for generating CPU loading times across the whole set of jobs. Any other standard probability distribution function could be used with appropriate parameters.
- ITERATIONS – The number of GridLoader jobs to create
- NEXTREQ_[MIN/MAX] – Used in a simple batch scheduling script, defines the range of wait times before submitting the next job. The values are normally distributed within the set range.
- NEXT_HOST_[MIN/MAX/PREFIX] – Also used in simple batch scheduling operation, defines the next host's IP address to which the job will be submitted.
- The following parameters define the ranges for the generation of parameters influencing the behaviour of a single GridLoader instance on the node it is executing:
- CPU_LOAD_PARETO_B_[MIN/MAX] – Sets the upper and lower bounds on the Pareto B parameter; range of values is generated using normal PDF.
- IP_[LOW/HIGH/PREFIX] – Defines the range of IP values for the target IP address of the GridLoader network peer. Could be defined as a single IP address to simulate a master-slave Grid environment.
- MEM_[MEAN/MIN] – Sets the GridLoader's memory allocation parameter. The value is calculated by adding a random number with the mean of MEM_MEAN to the minimum value defined in MEM_MIN.
- NET_[MEAN/MIN] – Sets the GridLoader's network transfer time parameter. Calculated in the same way as the memory value above.
- BURST_[MEAN/MIN] – Sets the GridLoader's inter-packet delay parameter. Calculated in the same way as the memory value above.

5.3.3. Deployment Scripts

The deployment application will generate a file containing appropriate parameters for each GridLoader instance, and a configuration file for the batch scheduling script. The probabilistic nature of the GridLoader is here evident at different levels. At the global level, two job sets with the same parameters will not have the same single values, but in both cases those values will fit the same, requested, statistical distribution function. At the level of a single GridLoader instance, two equally parameterised runs on the same machine will adhere to the parameters supplied, but will achieve those targets with a different resource utilisation profile.

To help visualise the job set being run, deployment application produces a plot of parameter values with relevant histograms, as shown in Figure 8.

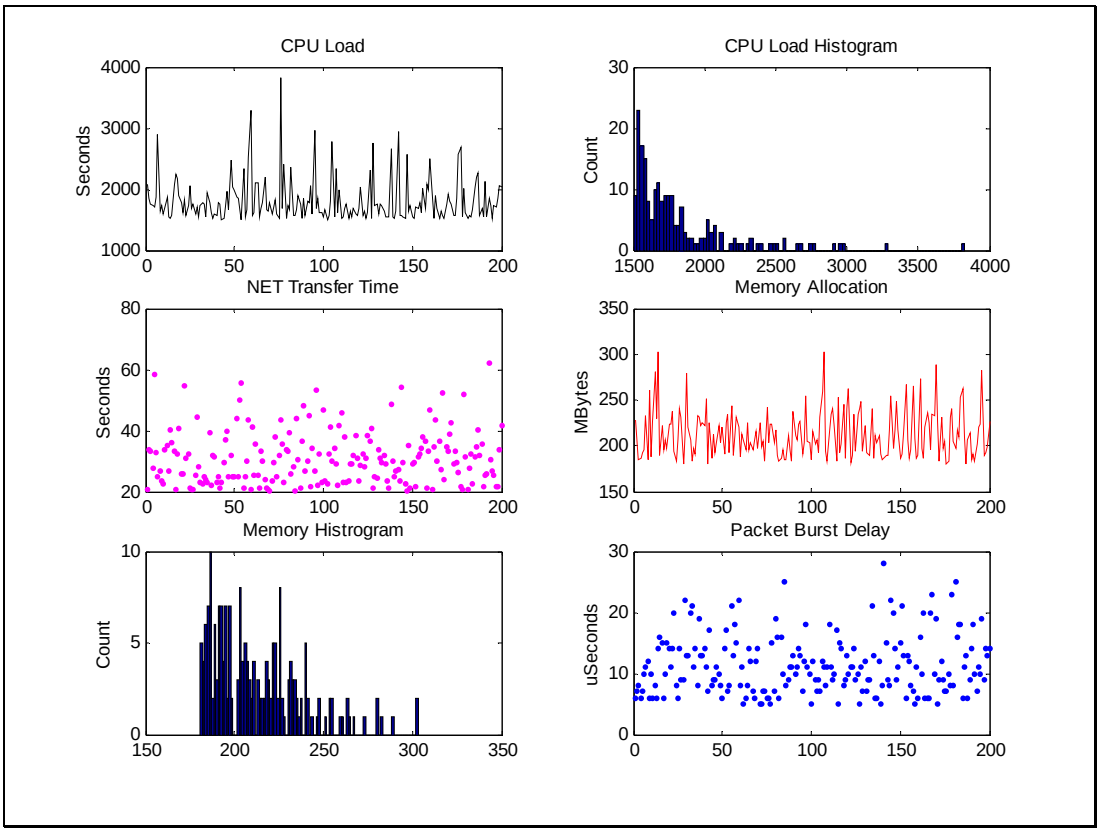


Figure 8 Matlab Generated GridLoader configuration file output

5.4. Self-Test Results

Before using GridLoader to test other components of the SO-GRM management architecture, a test of its own reliability was undertaken. Primary concern was the quality of resource utilisation models and adherence to the specified parameters such as the execution time and the size of memory allocated.

To test the reliability of overall timekeeping, a set containing 120 jobs taking around 24 hours to complete was created and run in sequence on one node in the testbed Grid (see Figure 4). A simple batch scheduler script was run on a "master" node and used to submit jobs through either Globus Toolkit 2.2 middleware or Secure Shell (SSH) to a dedicated "slave" node. Same job set was then re-run locally on the "slave" machine in order to differentiate between GridLoader's systematic error and any overheads that these middleware introduce. Figure 9 shows a percentage difference between expected and actual execution times for a sample of 50 jobs and for all three different execution methods.

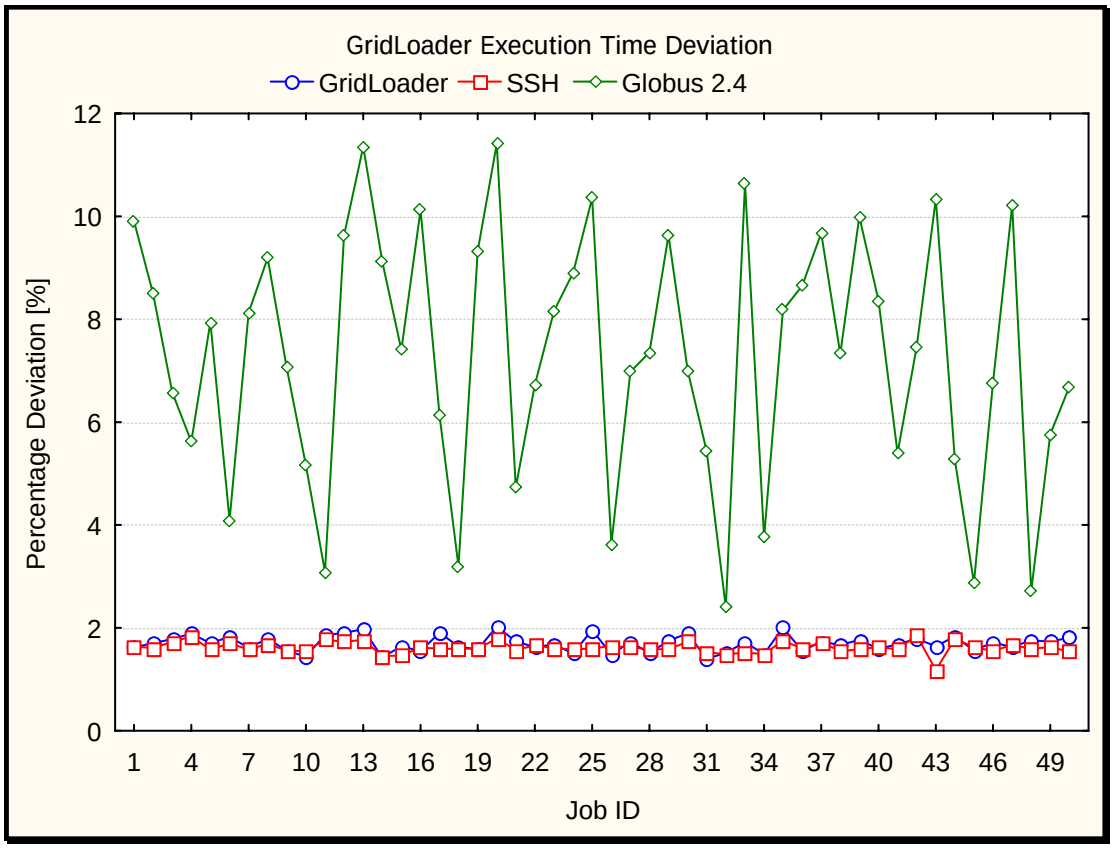


Figure 9 Expected v Actual GridLoader Runtime

Running on the local node, actual GridLoader execution times are less then 2% greater than expected. This is due to the system overheads such as setting up network transfers, allocating memory and random number generation, which are not accounted for in the timekeeping of the program. As this level of increase in execution time is intrinsic to the operating system, and would be present for all applications, we found that a realistic and accurate simulation of the total length of the job can be achieved using GridLoader.

As previously described, a loose control on the level and shape of CPU loading can be exercised by specifying different values of Pareto parameter B at run time. A parameter sweep test was undertaken to establish the upper and lower bounds of these values that provide a usable result. During these tests it was noted that a low value of the parameter will result in a longer duration of CPU idle time, and thus a lower average load. Higher values of the shaping parameter cause Pareto probability function to draw high numbers for CPU intensive loops and leads to higher average utilisation and pronounced load spikes. GridLoader's probabilistic routines will create a similar, but not equal, trace for each equally parameterised run.

Reliability of the duration of network transfers was established as part of the overall test of GridLoader timekeeping. The influence of inter-packet delay parameter was examined through a parameter sweep test. By using network monitoring package Iperf, bandwidth utilisation between the "slave" node executing GridLoader and a designated traffic sink node was measured. The inter-packet delay parameter provides a soft control of the amount of bandwidth

used, and not a strict upper or lower limit. This kind of probabilistic behaviour is sufficient for the required simulation of the network traffic and, considering the aims of the simulation, its probabilistic nature is beneficial. The use of the UDP network protocol, and its lack of bandwidth control mechanisms, could lead to network congestion issues in large GridLoader simulation runs. It remains to be assessed whether such conditions would impair the running of the simulation or added another realistic aspect of the production network environment.

Sequential memory allocation and freeing has been monitored using the Ganglia system, as shown in Figure 10. The test were carried out to confirm actual physical memory is being allocated, and that this could lead to memory contention as in the case in production environments. The granularity of the allocations is one megabyte which is considered sufficient considering very high memory utilisation of most Grid applications.

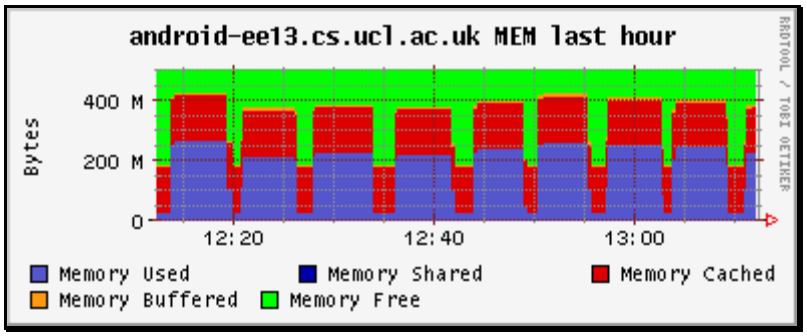


Figure 10 GridLoader Memory Utilisation Testing

5.5. Conclusions

In this chapter, a novel Grid application load simulator tool, GridLoader, has been described. GridLoader provides a way for parameterised and probabilistic simulation of application CPU, memory and network usage. Deployment scripts facilitate creation of run-time parameters for large simulation runs, enabling these to follow statistics of jobs observed on the production Grid facilities. Testing of GridLoader functionally and reliability has been undertaken and reported on.

During the stand-alone testing phase of the GridLoader, a number of minor problems and issues were discovered.

From the implementation perspective, a better CPU loading algorithm would prove very useful. Some cases exist where a constant, predefined level of CPU load should be simulated, such as visualisation applications or other applications bound not computationally but by some other factor. These could not be precisely simulated using the currently implemented probabilistic approach.

GridLoader heavily depends on the quality of the random numbers generated within the programme, and the seeding mechanism for the random number generator. Although better generators than the one used in GridLoader are available, these would require additional libraries which may not readily be

available on the target platforms. As no adverse effects associated with random number generations were observed during debug runs, the current approach is considered adequate.

Significant problems were caused by the real-time clock resolution and lack of synchronisation between the Grid nodes. Globus X.509 certificates carry a begin-end validity period with one second granularity, and in a network without proper clock synchronisation a certificate may become valid on one machine before it does so on another. This leads to the job being rejected due to incorrect credentials, an error message often associated with other issues within the Globus Security Infrastructure and Certification Authority problems.

Overall, the parameter generator application and the GridLoader were successful in creating a job set with given statistics, and executing it according to the parameters required. Appropriately parameterised GridLoader will be able to simulate a realistic eco-system of Grid applications and present a diverse and varied load to the Grid management components on test.

Development of GridLoader is a distinct contribution of this thesis. Apart from its primary intended use as a Grid application simulator described above, GridLoader can potentially be used as a testing tool for confirming end-to-end application level operation of Grid middleware. With a suitable parameter set, GridLoader could also be used to stress Grid hardware and middleware components to the edge of their operational envelope, thus exposing any possible points of failure or performance bottlenecks.

6. CHAPTER SIX

MONITORING FRAMEWORK

A detailed account of the monitoring framework developed by the author will be presented in this chapter. Section 6.1gives the motivation for developing the monitoring suite, while Section 6.2 captures the system requirements. Section 6.3 gives details of the implementation, Section 6.4 provides results of functionality and reliability tests, while Section 6.5 concludes the chapter.

6.1. Motivation

Current Grid monitoring systems, as previously summarised in Section 3.3, offer scalable and effective monitoring of resource utilisation on a per-node basis. As one must assume a general case where Grid nodes will be both time- and space-shared, these measurements bear no relationship to the actual resources used by any single application. Even in the case of a dedicated Grid host, the footprint of current Grid middleware, management and security components is such that the overall node resource utilisation will be very different to that of a single user application.

The author's motivation was to extend one of the current monitoring systems to provide process-specific measurements of resource utilisation in an unobtrusive and scalable way. Extension to an already established monitoring system would have the benefit of an already established user base, giving access to wider source of data. It will also remove any switching cost from the user's perspective and alleviate administrator's reservations about installing unproven software.

6.2. Requirements

The basic requirements for a Grid monitoring system are support for a wide range of operating systems and hardware architectures, effective data storage methods, and the use of efficient and standardised communication protocols.

The additional requirements for successful integration with other SO-GRM management components were an extensible metric sampling interface, the possibility of integration with the Globus MDS, and support for XML encoded messages.

The monitoring system of choice should be able to integrate per-process resource utilisation metrics into the standard flow of measurement data, fully supporting storing and retrieving such additional information through its usual data access methods.

6.3. Implementation

The measurement suite herein presented is based on the Ganglia cluster monitoring system. Ganglia was selected for its extensible data collection interface, effective storage of data in fixed size round-robin databases, use of XML encoded measurements, and customisable unicast and multicast delivery protocols. It has previously been extensively used with Globus Toolkit and successfully integrated with MDS using the Glue Schema [83]. Various platform-specific information providers have been developed, and this modular design offers a clear path for implementation of per-process resource utilisation monitoring.

6.3.1. Ganglia Functionality

The monitoring suite is implemented through a set of Ganglia applications, compiled code, and shell scripts developed by the author. All code was written with portability in mind and relies on UNIX standard libraries and script commands. Figure 11 presents the layout of monitoring components in a block diagram.

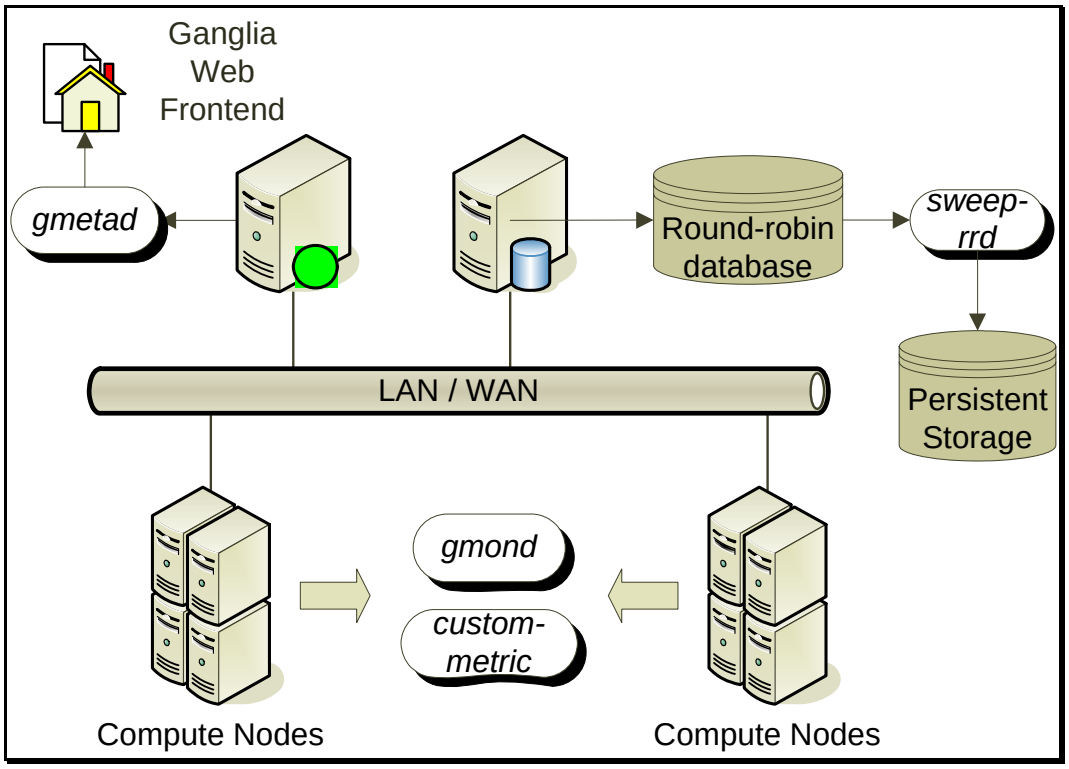


Figure 11 Monitoring Components Block Diagram

Ganglia Cluster Monitoring core provides two daemon modules:

- *Ganglia Monitoring Daemon (gmond)* – collecting basic information about each node in predefined intervals, encoding it in XML and providing network transport mechanism
- *Ganglia Meta Daemon (gmetad)* – receiving information broadcasted by all or some of the monitoring daemons, and storing it in the round-robin databases. It also answers queries about overall state of the cluster, and provides a programmatic interface to queries on data contained in the databases.

Round-robin database (RRD) [60] is a fixed sized database targeted at storing time-series data. Each database can contain several data sources (DS), and each data source has a number of round robin archives (RRA). These archives could be thought of as layered tooth-wheels, each wheel slot containing one sampled value. On database creation the frequency of rotation of these wheels is defined, and a consolidation function (CF) is given for each data source. Once the wheel makes a full turn all of its data is passed through the consolidation function (usually average, minimum or maximum) and the result is written as one sample point in the next hierarchical wheel. The size of the database is kept constant, since the high frequency data will be kept for a limited duration before being consolidated. Depending on the target application, this behaviour may be a desirable feature or a disadvantage.

Ganglia Monitoring Daemon can use either unicast or broadcast UDP packets to transport XML encoded measurements. Each *gmond* can be set up to either listen to other daemons (mute mode), transmit its measurements to other peers

(deaf mode), or do both. By configuring certain nodes to be muted or deafened, a distributed system with no single point of failure can be created. In our test implementation, all but one Ganglia monitoring daemons were configured in deaf mode. One node in the network run the non-deaf daemon, as well as *gmetad*, and provided storage for all databases. This centralised network configuration was appropriate provided the size of our test network (no more than 10 nodes at any time), and the goal of our tests.

6.3.2. Information Providers

The author has developed custom information providers to provide monitoring of CPU utilisation and memory footprint per each process submitted through Grid middleware. It is implemented either as a shell script using UNIX standard *ps* command, or as a precompiled application (still in development) using *libgtop* library. Functionality is similar, as both implementations run as a daemon on each Grid node and periodically sample CPU and memory utilisation. Criteria for process selection, and the information collected are fully customisable. Processes can be selected by their process identification (PID), executable name, or by username under whose credentials they are running. Information reported can include any metric available through UNIX */proc* system. Process selection based on the PID is the most efficient and unambiguous method, unfortunately current implementation of Globus Toolkit does not pass the PID of the remote process to the job scheduler, nor does it make this information available through MDS or any other means. This is a widely recognised implementation issue, impeding improvements in several areas such as grid job workflow and scheduling concurrency. Next versions of Globus Toolkit should address this problem. Once the per-process monitoring data is collected, it is transmitted either by using Ganglia's *gmetric* shell command or by using Ganglia's API libraries, depending on the implementation.

6.3.3. Database Management Tools

Author's core research topic will be based on the initial assessment of the monitoring data, and the development of a suitable prediction algorithm, both of which will depend on the availability of large amounts of high frequency data. Consolidation feature of round-robin databases is therefore not beneficial, as high frequency data would be quickly lost through averaging. A shell script, *sweeprrd* in Figure 11, was developed to perform automated data extraction from RRD databases. The script can be configured to retrieve data on specific nodes and specific metrics of those nodes, or collect all the data available. Time stamped measurement values are formatted in a comma delimited format, and stored as flat text files. The script can either be run as a daemon process or invoked by UNIX standard *cron* scheduling daemon. Frequency of execution is customisable with the obvious limit of at least one sweep within the duration of the shortest round robin archive in the database (to prevent any data being lost through consolidation). Database sweeps can be invoked as often as necessary and at any time; the script will only extract new samples from the RRD database.

6.4. Test results

First phase of the monitoring suite tests was aimed at establishing proper installation and basic functionality of the Ganglia suite. After modifications to Ganglia's default settings, it was necessary to ensure core functionality has not been affected and stable operation was maintained. Ganglia version 2.6 was deployed on both BT and UCL administrative domains of our testbed Grid (shown in Figure 4, and used for day-to-day monitoring of these facilities. Figure 12 shows a typical screenshot of Ganglia web front-end displaying overview of hosts in the UCL domain.

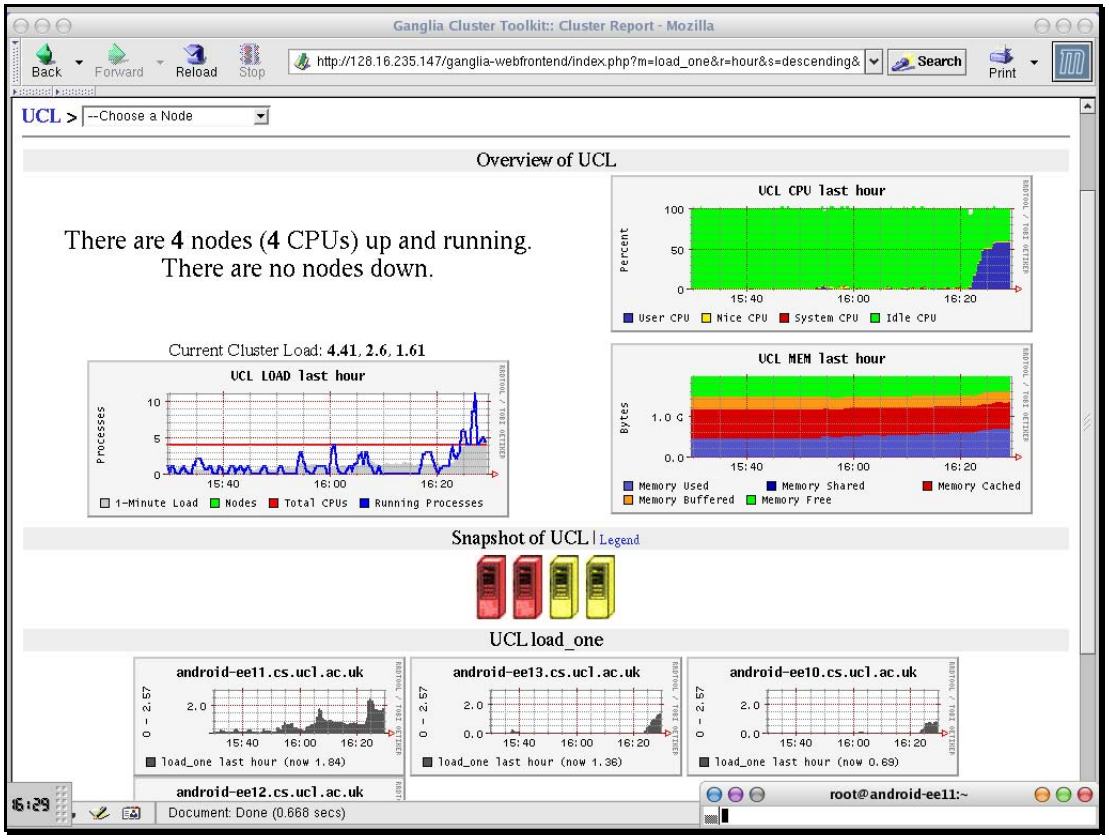


Figure 12 Ganglia Screen Shot - Cluster Level

In the second phase of testing, per-process monitoring components were introduced and observations were made on stability of the system, quality and reliability of measurement, and any increase in system resources utilisation. Screenshot in Figure 13 shows a single monitored node in the Grid under heavy utilisation, while screen detail in shows *globus-cpu-utilisation* metric, revealing the CPU utilisation attributed to a single Globus submitted job.

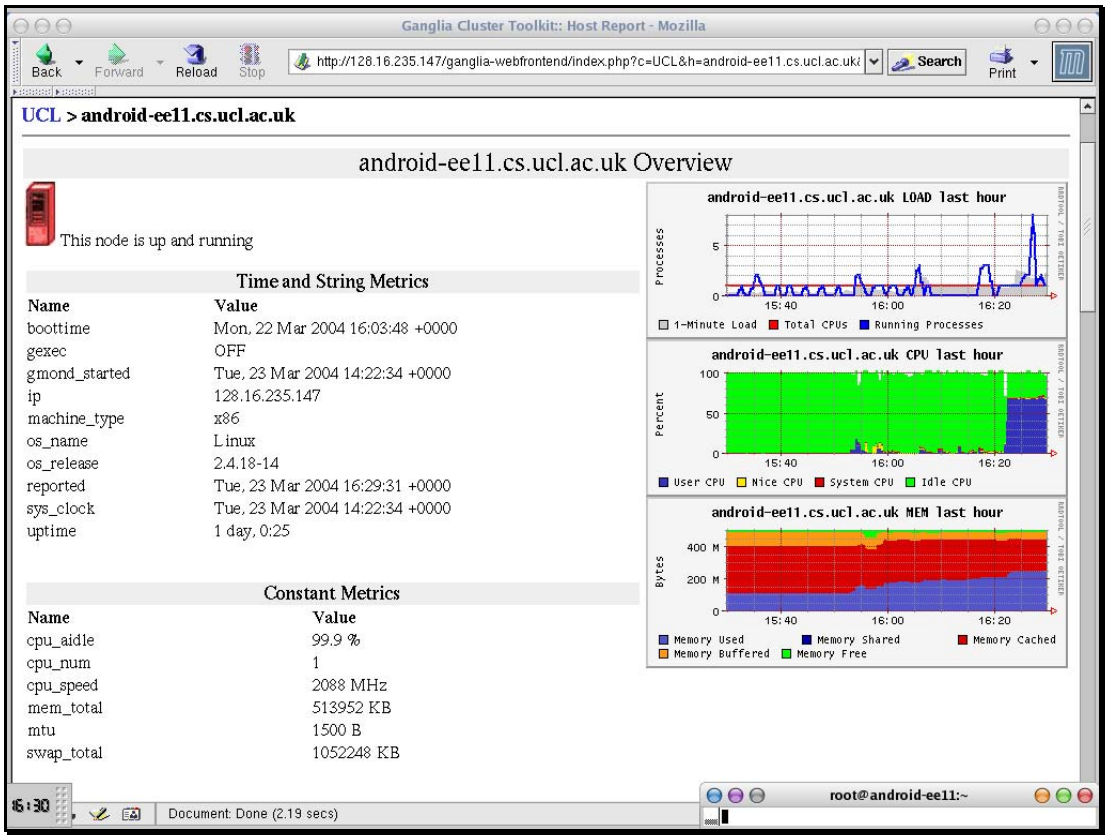


Figure 13 Ganglia Screenshot – Node Level

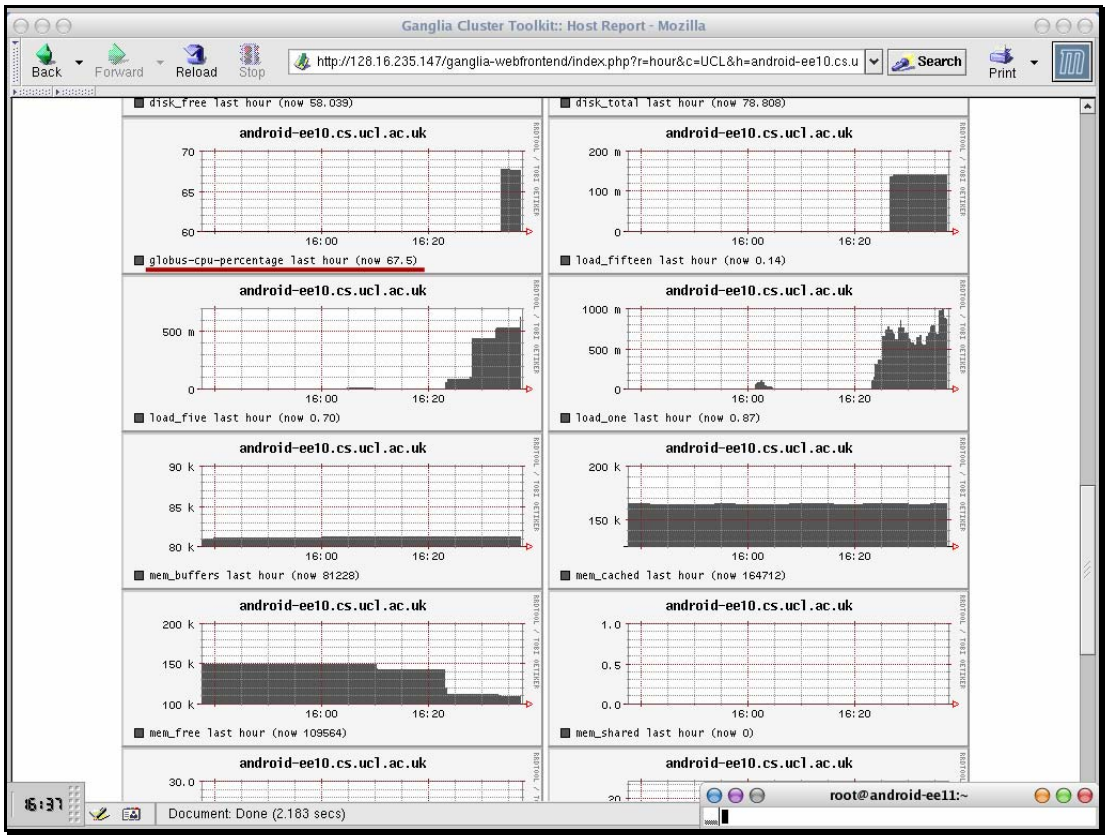


Figure 14 Ganglia Screenshot – Custom Information Provider (Highlighted)

The third phase of tests was designed to establish the overall monitoring functionality and the quality of measurements. A sample GridLoader set containing 50 jobs with Pareto distributed execution lengths was run on a single machine on the Grid testbed. Full set of metrics including Globus-attributed and total CPU utilisation were recorded through our monitoring suite with one second resolution, averaged and published over 15 second periods. Jobs were submitted from one of the machines in the cluster to a different machine in the same cluster using appropriate Globus commands. A simple master-slave scheduling was used, iterating through the job list and allowing 45 seconds between job completion and next job submission for any transient machine loading to settle. These transients loads are created by the Globus toolkit job completion procedures (results stage-out, process cleanup and accounting updates). Resulting data is plotted in Figure 15.

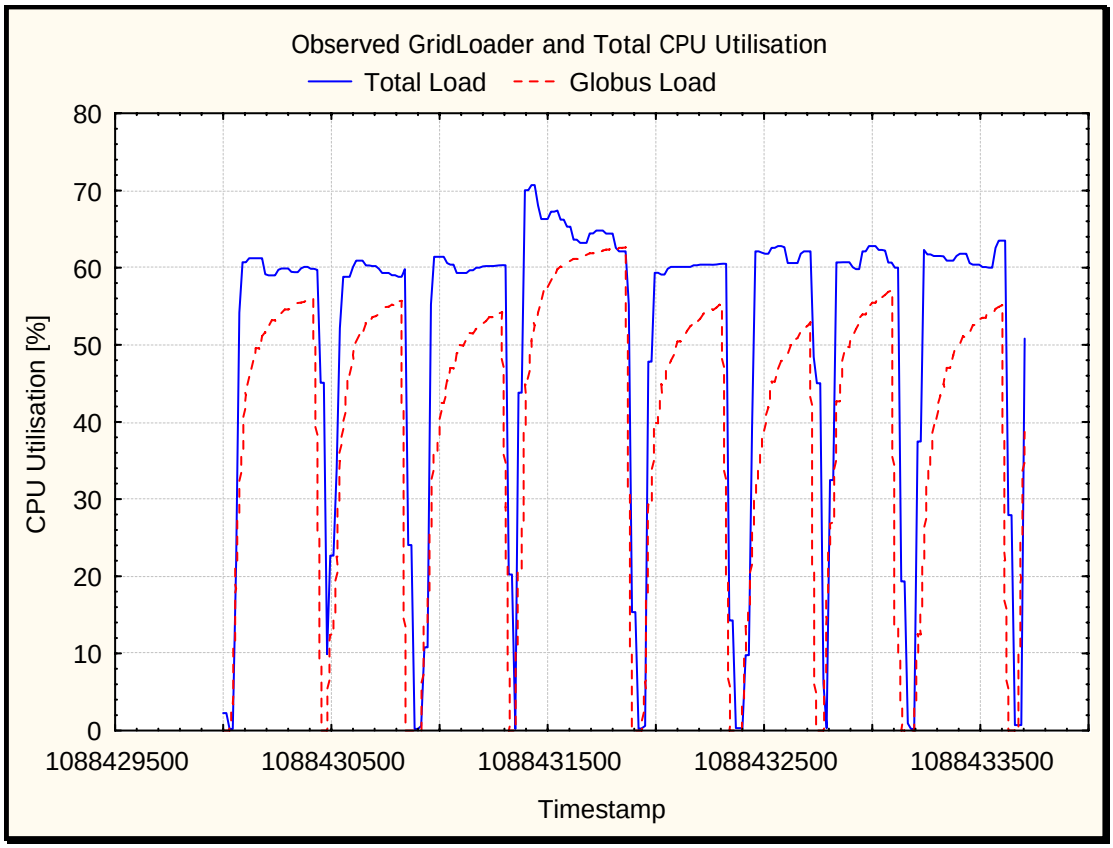


Figure 15 Comparison of GridLoader and total CPU utilisation

The measurements captured the difference between the GridLoader generated load and the total system load which includes various background processes associated with Globus middleware, and kernel time servicing network transfers, memory allocation and process scheduling. Figure 16 shows in more detail percentage difference between total and GridLoader CPU utilisation. Positive values indicate greater reported system load than GridLoader generated load.

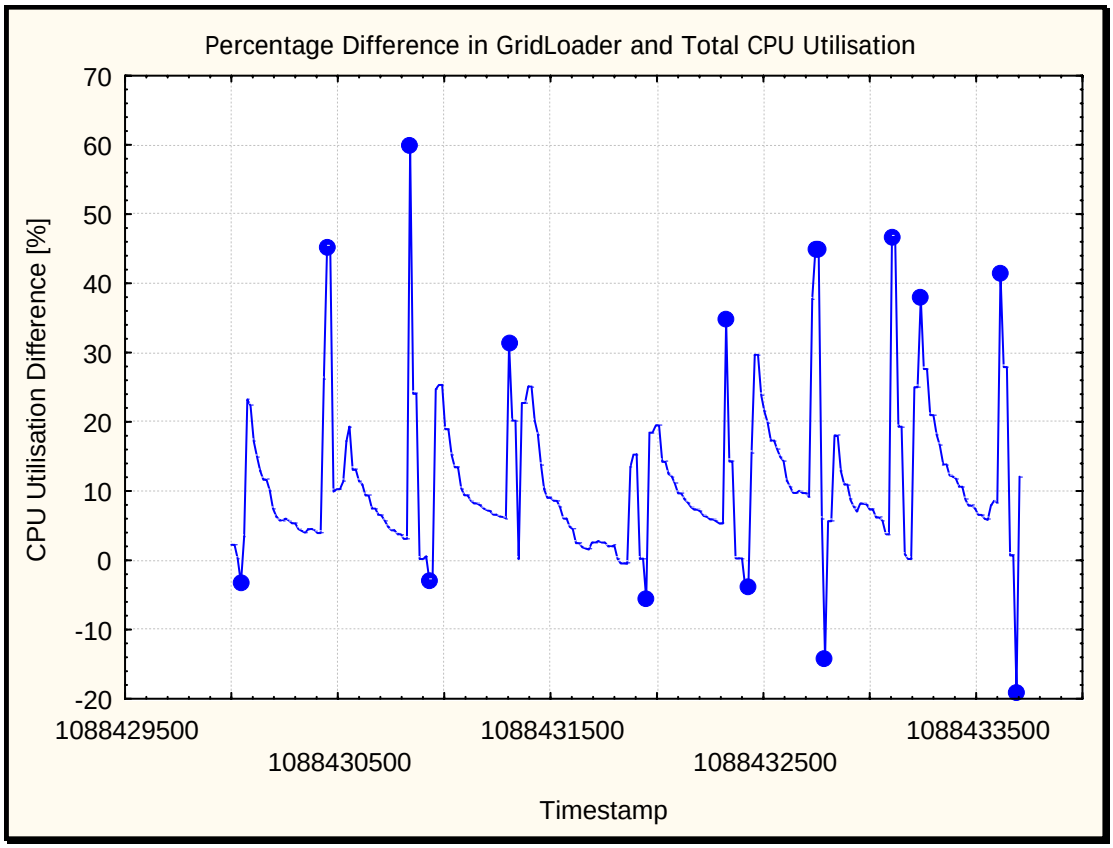


Figure 16 Total and Globus-attributed CPU Load Discrepancy

The data shows discernible and repeated peaking in both positive and negative values. Markers on the same plot indicate recorded job start and end times and there is a visible correlation between these. Analysis of job events at these timestamps indicate that discrepancies are partly attributed to the submission and staging of the next job in the queue. At those times, the machine loading is high, but the CPU time is not yet attributed to the process being submitted. The samples coinciding with job start and end times can be filtered, which will lead to a more balanced plot as shown in Figure 17.

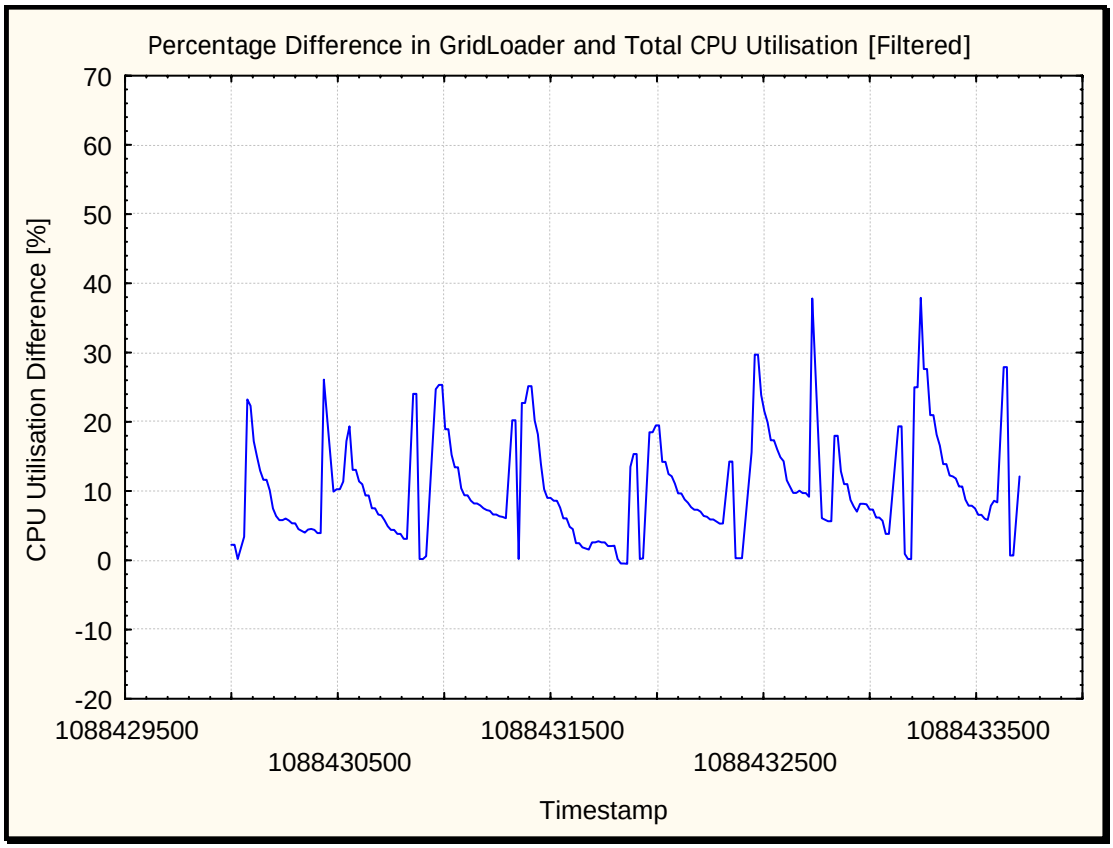


Figure 17 Total and Globus-attributed CPU Load Discrepancy [Filtered]

This experimental data has exposed a shortcoming in the process monitoring component which leads to a ramp-up effect in the observed loading measurements, as seen in Figure 15. This low-pass effect causes large variation between total node utilisation value and Globus-attributed CPU load at the beginning of job execution. The software routine responsible for collecting those measurements uses UNIX standard process reporting calls, and these return CPU usage as a decaying time average since process initiation. To improve the accuracy of measurements, a trade-off will have to be made in the portability of the code. Work on an improved version is under way, focusing on the use of kernel "jiffies" measurements for a more reliable result.

Another source of discrepancies is different sampling frequency of per-process and total CPU load measurements. This leads to discrepancies at the end of job execution. In those instances, per-process load would disappear before total load is reduced, and this time offset would lead to large difference being reported.

Most of the problems occurred where in the local monitoring component. However, successful overall operation of the system was confirmed, and sampled data was correctly integrated in the Ganglia data handling flow (including Web visualisation). Data extraction tools operated effectively and reliably with no lost or duplicated samples. Data obtained was readily analysable, and has immediately provided insight into the extent of difference between perceived and actual resource usage by Grid processes.

Resource footprint of the monitoring components was acceptable (below 1%); although an increase was noted as the number of processes to be monitored grew.

This is attributed to the computationally expensive parsing of the processes table required to obtain process IDs of monitored jobs, and depends strongly on the criteria used for process selection.

6.5. Conclusions

Presented monitoring solution addresses the whole monitoring cycle, from measurement data collection, to visualisation and extraction for off-line analysis. The system has been developed on an open framework to support programmatic access to the data by the forthcoming scheduler component. Implementation has taken into account expressed reservations of cluster administrators to running third party compiled daemons on their networks, and has developed a transparent monitoring system based on a widely used application. The chosen approach scales well, being based on a proven core and complemented with maintenance scripts designed to facilitate deployment and management. This solution seamlessly integrates measurements specific to the needs of the advanced scheduler research with an established monitoring framework. Off-line data analysis is facilitated with the use of the data extraction scripts developed.

Captured measurements, combined with the accounting data from a production Grid cluster, will allow analysis of the job statistics such as arrival times, queue wait times, resource utilisation and execution times. It should be possible to locate any correlation between multiple job runs, and some social and workflow factors such as time of the day or day of the week when the job was submitted, and user identity. Considering human workflow, one intuitively expects that a certain user working on a project will submit jobs of similar nature at similar times of day expecting a certain turnaround time. Recognising such localities in the covariant data, and communicating them to the scheduler could result in an increase in the quality of its predictions.

7. CHAPTER SEVEN

TOWARDS A PROBABILISTIC SCHEDULER

Examining schedulers currently in use on the Grid (see Section 3.2), it is clear that current systems leave much to be desired in terms of flexibility, user convenience and scheduling efficiency. If the Grid applications are to be successfully mapped onto highly dynamic Grid resources a break from the legacy batch systems seems inevitable.

This chapter will introduce the envisaged probabilistic scheduling approach, and is organised as follows: Section 7.1 gives overall aims of this novel scheduling approach, Section 7.2 presents methods that will be investigated for delivering such scheduling, while Section 7.2 captures the requirements and restrictions of the future system. Preliminary data analysis is presented in Section 7.4 and the deployment plan is given in Section **Error! Reference source not found.**. A brief summary and conclusions are given in Section 7.5.

7.1. Aims

To truly live up to the utility computing vision the Grid has offered, application scheduling needs to offer to the user a decoupled and service orientated environment, requiring no in-depth knowledge of the physical Grid resources or application characteristics. The ultimate workflow experienced by the end-user should be similar to a dry-cleaner service or a photo developing lab: the user presents a job and decides between several, differently priced, turnaround time options. The service provider can then decide on how and where to service the

request, trading off between the cost of scheduling (perhaps sending photos to be developed off-site) and the cost of local resources.

Achieving this level of user abstraction from the underlying scheduling process requires good knowledge of the nature and complexity of the jobs submitted. Since these are hard to capture, and may be unknown even to the end-user, the scheduling system will inevitably need to make predictions and rely on estimates.

The author's ongoing research will focus on the investigation and development of the algorithms estimating the job's execution time and its resource utilisation profile. The aim of these predictions is to enable a probabilistic scheduling approach, one that removes hard limiting in terms of execution time or resource utilisation, and instead relies on the diversity and statistical distribution of Grid application to ensure adequate scheduling.

Such approach would benefit from manageable oversubscription by statistically multiplexing jobs at the VO level. From the experience of running batch schedulers – which has shown that users tend to grossly over-estimate the requirements of their jobs – this approach could lead to increased overall utilisation levels while ensuring user's quality of service is unaffected. This is a key added value proposition for present hardware operators which are dimensioning their clusters for peak load and thus seeing very low mean utilisation levels. A strong business case exists for a novel scheduling technology that will reduce the mean-to-peak load while maintaining user's perceived QoS.

7.2. Requirements

As previously discussed in Section 2.2, a novel scheduling system would need to offer added value to the end-user, the Grid resource owners and, indirectly through the reduction of the total cost of ownership, to the Grid administrators.

From a user perspective the job submission should be simple and the switching cost minimal – the new system should not require recompilation of the source code or require the use of any specific middleware.

Resource owners are primarily concerned with maintaining high overall utilisation, and can benefit greatly from the statistical multiplexing and oversubscription that probabilistic scheduling could deliver. A simple control mechanism should be made available allowing the overall level of utilisation and the level of the scheduling safety margin, to be easily and dynamically changed. Through such mechanisms, resource owners should be able to trade-off resource utilisation levels with the quality of service provided according to their business model.

By using commodity components in the Grid clusters, the total cost of operation is significantly influenced by ongoing administration and maintenance expense. New middleware components would need to ensure administrative burden is kept to a minimum by trying to offer a self-managed service able to adapt and autonomously resolve as many operational issues as possible.

Proposed scheduler will make no presumptions of the nature of the jobs running, their monolithic or distributed nature, criticality of any one sub-system on their operation (network performance, memory bandwidth and similar), or their internal optimisation for any given platform. Rather than try to model these parameters directly, it is expected that their influence will reflect on one of the monitored metrics. For example network performance may strongly influence the execution time of a certain job, and may also be very dependant on the time of the day when the congestion is most likely to occur. The scheduler may therefore pick up on the correlation between submission time and actual execution time for a certain job highly dependant on network bandwidth.

7.3. Methodology

To estimate a job's execution time, the author's selected approach will investigate the relationship between an application's historical execution times and meta-data related to the operating environment at the time of execution. A range of "hard metrics" such as scheduling wait time, execution time, effective CPU time, memory utilisation and similar will be investigated. These will be observed in a broader perspective of the Grid environment, correlating them with a set of various "soft metrics" such as job submission time of the day and day of the week, submitting user and group, overall load of the cluster etc.

Intuition, supported by preliminary analysis presented in the following section, and the sequential nature of the human workflow leads me to believe that an observable and predictable utilisation pattern will develop in a production cluster with a significantly large and diverse user group.

To benefit from these, the proposed scheduler should enable autonomous and intelligent coupling of monitoring data, accounting records and operational meta-data in a framework able to analyse current and historical usage patterns of the system. It should operate with little or no user intervention, and act as a passive observer of the system's performance, user's habits and overall trends. Targeted at large production Grids, this probabilistic scheduling will be based on the statistics of a large number of jobs with widely varying resource utilisation, execution length and arrival characteristics. In this Grid eco-system, the scheduler should be able to distinguish trends, seasonal variations and inter-metric dependencies, providing a confidence factor for its predictions and learning on its own mistakes.

With a low system utilisation overhead, and negligible user switching costs, the scheduler aims to provide a better-than-guess estimate on job resource requirements and execution times. Although the accuracy of such predictions would be seen as a natural key performance indicator, previous research work and experiential evidence has shown that the timeliness of the predictions and the ability to subsequently refine them play a much more important role. In view of the dynamic characteristic of the Grid resources, an in the author's own view,

accuracy of the predictions will come second to the ability of the scheduler to constantly adapt to the changing operational environment.

7.4. Preliminary Analysis

Preliminary analysis of accounting and usage records of the UCL's Central Computing Cluster (CCC) facility has shown that a diverse user community had developed, running jobs with very different statistical properties. Based on accounting records since 10th of August 2004., an up to the end of year 2004., more than 50,000 jobs have been run totalling in excess of 5,500 CPU days.

7.4.1. Overall Job Statistics

The following analysis is based on a representative sample of 1000 jobs submitted in a 15 hour period in November 2004. Figure 18 is a log-normal plot of the wall-clock execution times of these. On its own, this series of execution times offers little insight into the effectiveness or quality of the scheduling, the level of resource utilisation, nor does it readily offer any predictions on the future loading of the system.

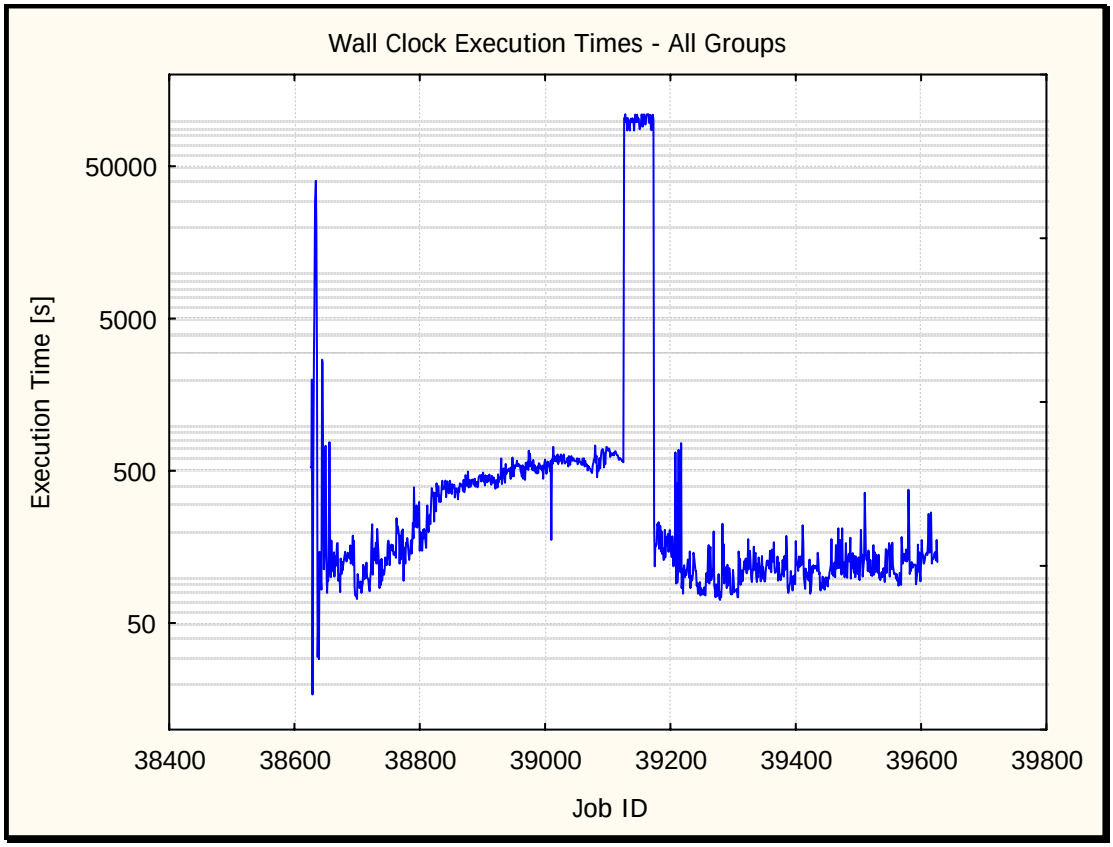


Figure 18 Wall-Clock Execution Times for All Jobs

However, Sun Grid Engine maintains a text accounting file containing extensive details of the jobs run in the past, including the username of the submitting entity

and the group to which this username belongs. On the CCC facility, these are centrally administered entries, set on account opening according to the user's project and department affiliations. Although the location of this data may change from system to system, it will inevitably be present on the accounting file in some way. Those fields can hint at the nature of jobs submitted by the users, and such job's resource requirements, execution times and arrival rates. Figure 19 is a pie-chart plot of the number of jobs submitted by each user group. Clearly a large difference between the groups exists, and further analysis should investigate the properties of each individual group.

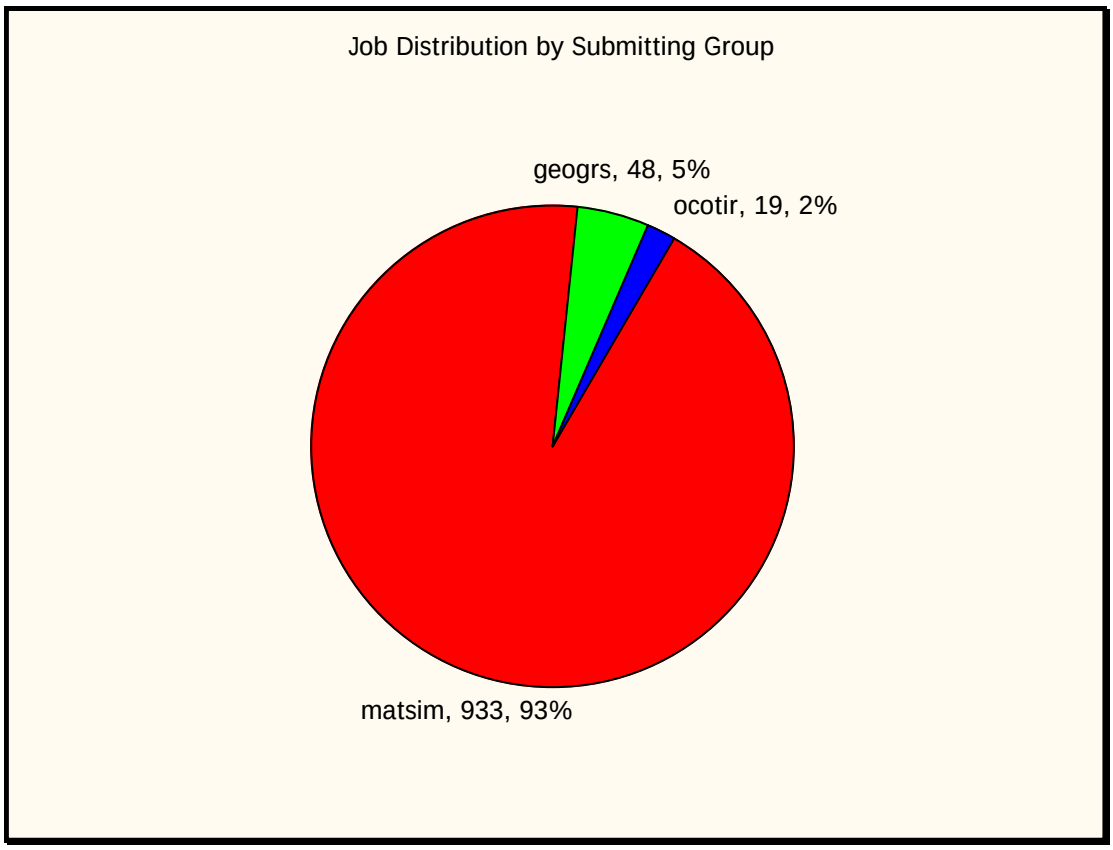


Figure 19 Job Distribution by Submitting Group

7.4.2. Group-based Job Differentiation

Separating execution times of each of the user groups, and graphing them as a time series plot in Figure 20, reveals a strong differentiation between groups.

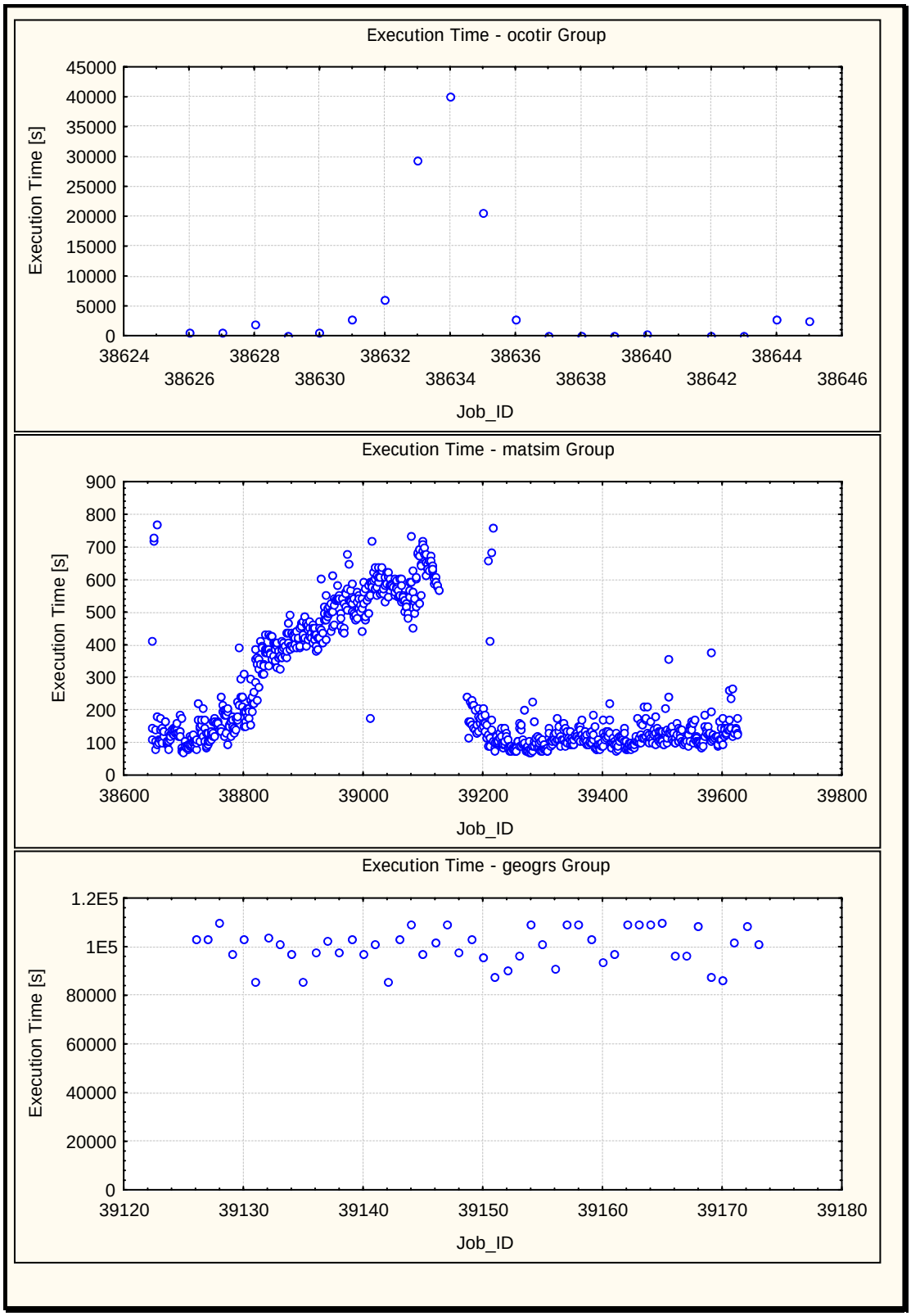


Figure 20 Wall-Clock Execution Times for Different User Groups

It is clearly seen that certain groups (such as *geogrs*) submit jobs whose duration is, on average, several orders of magnitude longer than those of other groups (like the *matsim* group). The number of jobs submitted by each group is also inversely proportional to their typical execution length.

Of great interest is the distribution of job execution times within each of these identifiable groups. An array of histograms in Figure 21 shows significant differences in the shape of these distributions.

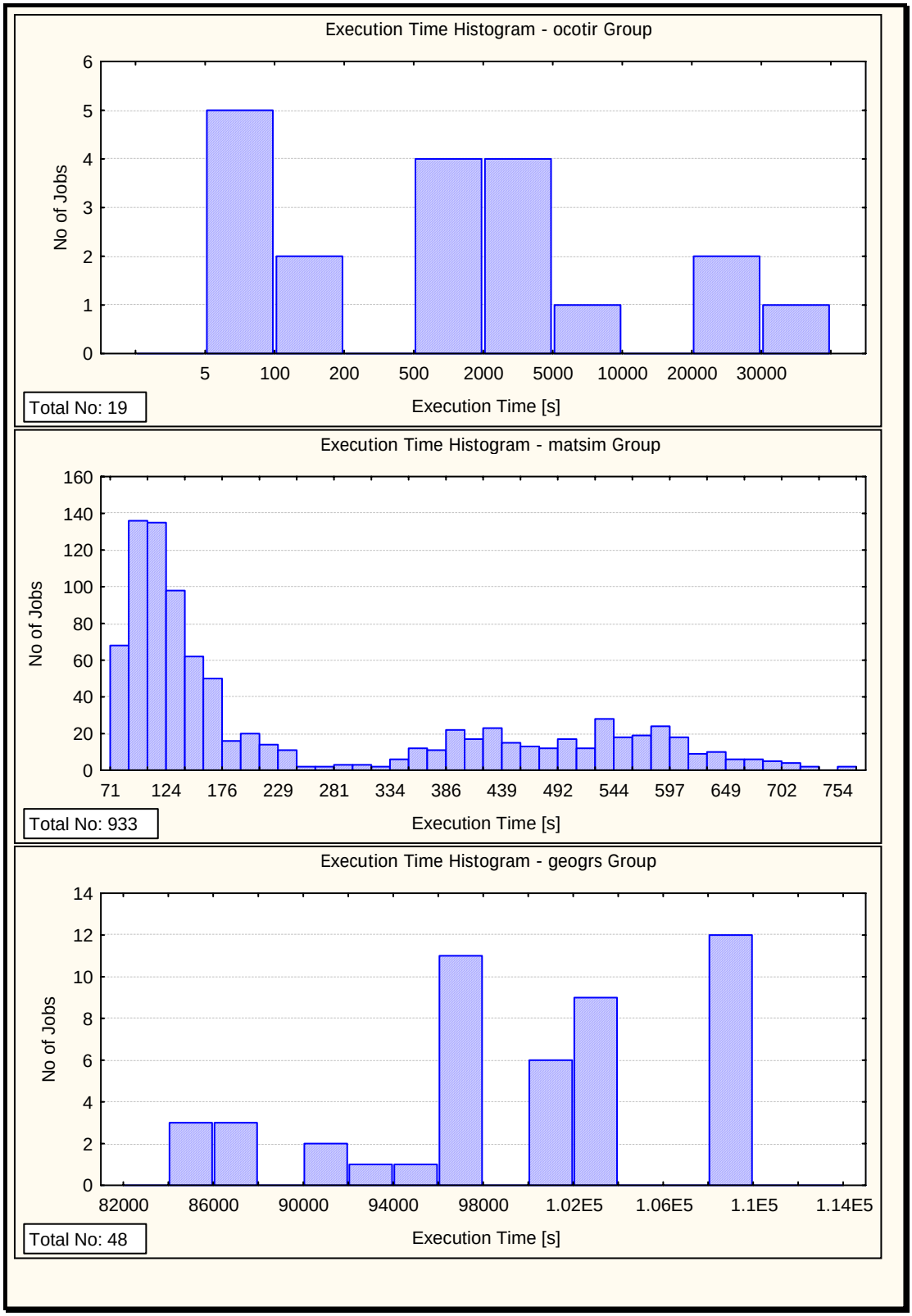


Figure 21 Histograms of Wall-Clock Execution Times for Each Group

Jobs submitted by some groups fall into normal distribution of execution times (like *geogrs*), while others follow a clearly skewed, long-tailed distribution. Applications will require development of different statistical models in order to represent them with acceptable accuracy. But benefit to scheduling process are already clearly visible: Figure 22 gives a mean plot with outliers (values 150% outside 25th - 75th percentile) of execution times of the three user groups represented in this sample of jobs (group isadmin had only one job and represents the system administrator account). The groups are clearly separated, spanning five degrees of magnitude, yet with a small 95% confidence interval relative to their respective execution times. These observations have important implications for the feasibility of predictions and their quality.

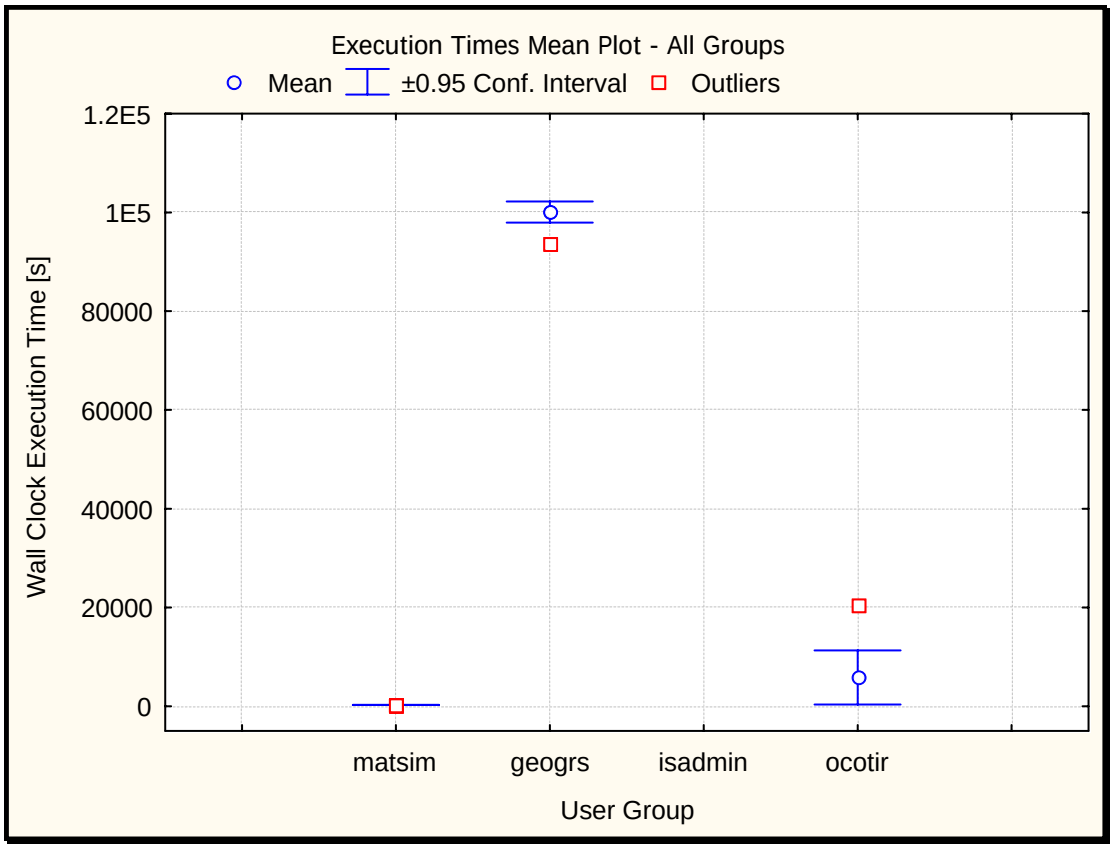


Figure 22 Execution Time Mean Plot for All User Group

Observing widely different means and confidence intervals plotted in Figure 22 reaffirms the need for a tuneable, statistical and probabilistic load generator, such as GridLoader, to adequately simulate observed resource utilisation of Grid applications.

7.4.3. *Data Clustering and Correlation*

Considering *matsim* group in more detail, see Figure 20(b), we observe at least two modalities of execution times, with a clear break at around Job ID 39150. Closer inspection of the raw traces reveals a 13 minute gap between two groups of *matsim* jobs. It is clear that some aspect of the application run by this group has changed as the application has started exhibiting a significantly different

execution time pattern. The scheduler will have to be able to recognise these changing temporal characteristics of the jobs, and appropriately adjust the prediction method and confidence level. By constantly monitoring the quality of predictions, the scheduler can decide to discard the previous "experience" if it proves to lead to greatly inaccurate estimates.

To effectively detect changes in metrics that can signal a new execution mode, the scheduler will need to analyse data for emergence of clusters. An application can swap between two or more clusters of execution times, or start executing in a new and yet undefined space. A preliminary analysis using K-means clustering algorithm[84] and three clusters have resulted in groups having the means as shown in Figure 23.

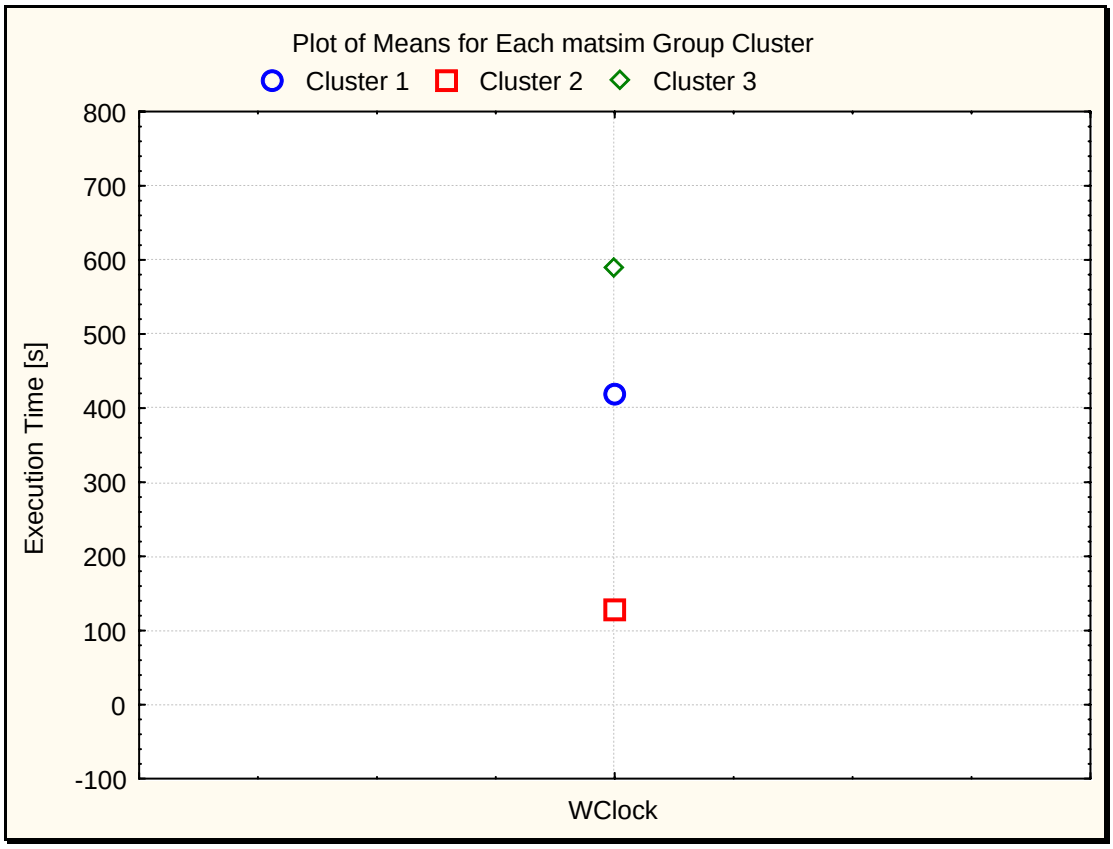


Figure 23 Plot of Means for Each *matsim* Group Cluster

Further details of the clustering analysis are given in Table 3. Cluster separation distances are given as distances below diagonal or squared distances above diagonal.

	Mean	Standard Deviation	Euclidian Distance		
			Cluster 1	Cluster 2	Cluster 3
Cluster 1	420	51.5	0	85214	29153
Cluster 2	128	38	292	0	214050
Cluster 3	590	56	171	463	0

Table 3 K-means Clustering Analysis

Selecting the first group of *matsim* jobs, with Job IDs in the range of 38646 to 39125, a study of their correlation was done. The level of execution time correlation can give insight into the predictability of the data and the reach of forward looking estimates possible. Figure 24 shows a partial autocorrelation function plot of wall clock execution times for said selection of *matsim* jobs with ten lag steps. The S.E. column in the figure represents white noise standard error. The execution times correlate with a significant degree up to seven lag steps, with a very high correlation present in the first lag step. Further correlation studies will play an important role in the selection of the prediction algorithm.

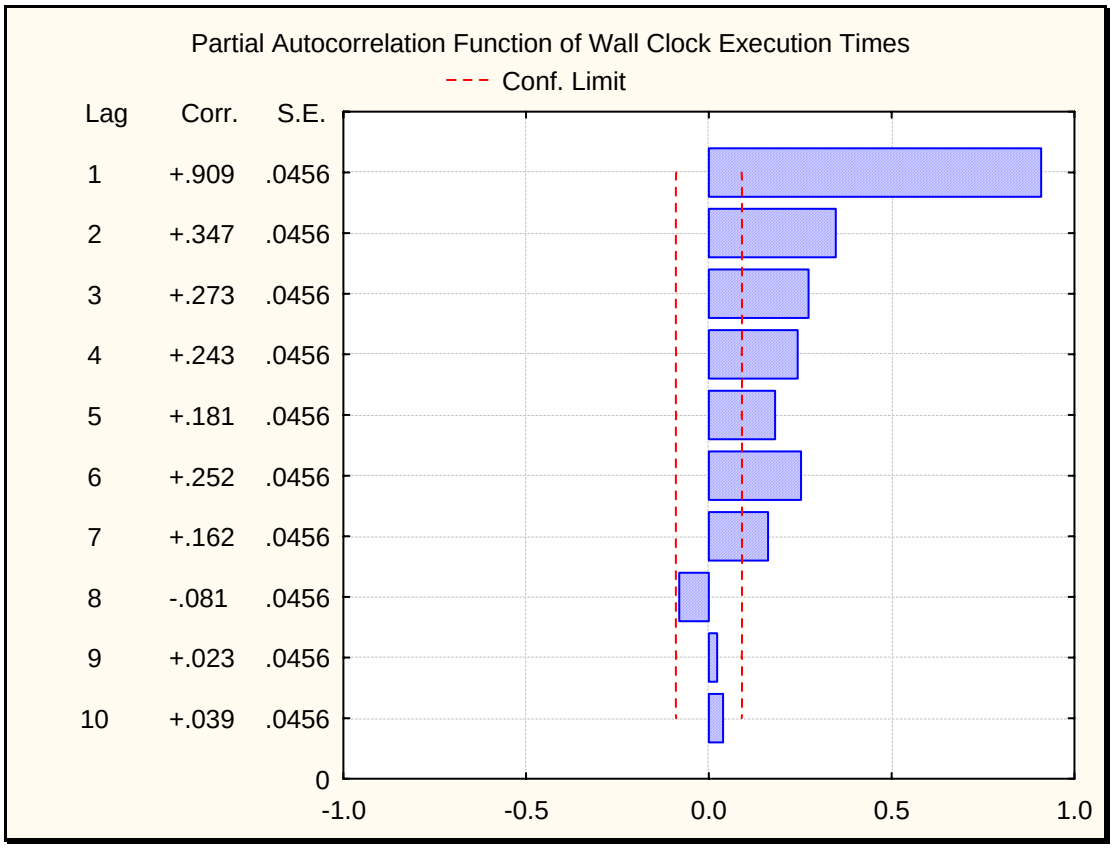


Figure 24 Correlation of Wall Clock Execution Times (10 lag steps)

7.4.4. *Temporal Characteristics*

Finally, the *matsim* job set was treated by simple curve fitting methods as an initial survey of its predictability. A distance weighted least square method was firstly used over the whole range of jobs. This method, as shown in Figure 25 in black dashed line, yielded a reasonable fit, but was marred by slow recovery from the abovementioned abrupt discontinuity of job execution times around Job ID 39150. A much better fit was achieved by using a negative exponential least square fitting function, shown in Figure 25 in red line, and this was further improved by using two fitting functions, one for each side of the discontinuity. Further research into these, and other forecasting methods will be undertaken.

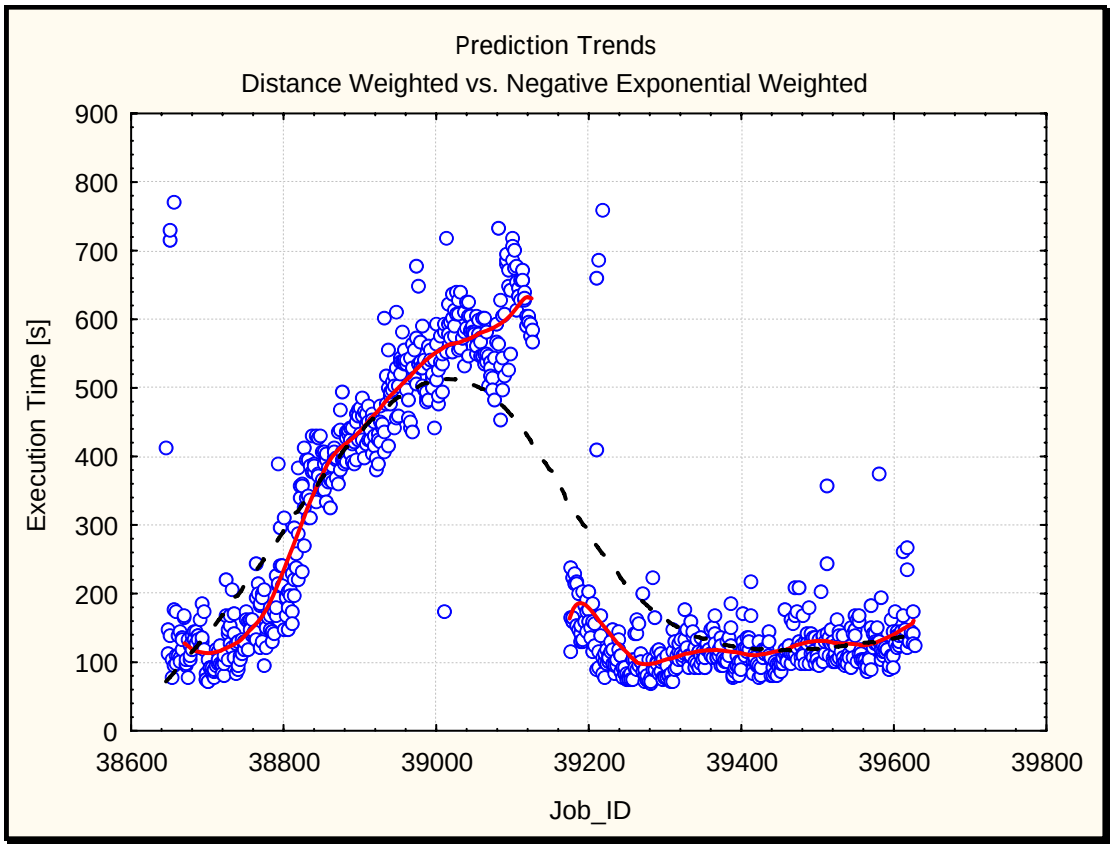


Figure 25 Curve-fitted *matsim* Jobs Scatterplot

This initial analysis of only two metrics, wall-clock execution time and submitting group, shows the possible benefits of statistical, stochastic and predictive scheduling system. In this example, on submission of a new job the scheduler can consult the historical information on previous jobs submitted by that user group, lookup the distribution of execution times and make a prediction on the length of the execution of the new job. Even a simple *if-then* test can distinguish between two different user groups and jobs that would, on average, run for 100s of seconds or 100,000s of seconds.

Important consideration in investigation of the collected data is its temporal characterisation. Following the human workflow, the type and the way applications are run on the Grid will change over time. Even from the relatively small amount of data in Figure 20(b), one can observe a positive trend in execution times for hundreds of runs, followed by a discontinuity with a large increase or decrease in subsequent runs. These abrupt changes may be caused by the change in the application's runtime parameters, update of the underlying data, or a new research objective may be identified. A robust scheduling system needs to be able to recognise these discontinuities and take steps to quickly adapt to new conditions, minimising the negative effects on prediction capabilities.

7.5. Conclusions

In this chapter, a probabilistic scheduling system for Grid environments has been proposed. This novel scheduling approach departs from the FIFO based queuing and instead offers out-of-order execution based on the job competition deadline, as set by the submitting user, and scheduler's forecasts on the execution length of the job and its resource utilisation.

From the user's perspective, such scheduling system could significantly simplify the job submission process, and enable a friendlier workflow by eliminating the need to explicitly state job's resource requirements.

For the Grid operator's, probabilistic scheduling could considerably increase the overall utilisation levels of their computational infrastructure by offering tuneable level of oversubscription, and by removing inherent "padding" of process requirements by the users present in the batch scheduling systems.

Job execution length and resource utilisation forecasts will be based on the historical data, and the meta-data collected from various sources relating to the submitted job, its owner, and the state of the Grid at the time of submission. Preliminary analysis of the data collected from the UCL's Grid facility shows that submitted jobs vary significantly in arrival rates, mean execution time and resource utilisation – creating a diverse application ecosystem which can support speculative and probabilistic scheduling technique as proposed.

8. CHAPTER EIGHT

FURTHER WORK

8.1. Algorithm Development

The development of the scheduler will proceed by further investigating data collected on the UCL's Central Computing Cluster (CCC) facility. The data will be analysed off-line with the aim of spotting statistically meaningful features and patterns, meta-data significant to the job statistics, and the trends and tendencies of job arrival rates and job workflows.

Theoretical work will centre on examination of various statistical methods for treating time series data, methods for analysis of such data, feature extraction techniques, and investigation of prediction algorithms. As a result of this off-line analysis, a set of suitable algorithms for on-line analysis will be defined and developed. These will enable integration of real-time monitoring data with the historical accounting information, providing for on-the-fly estimation of job execution times and resource utilisation of the newly submitted jobs. This is seen as crucial missing information in the scheduling systems such as ICENI (see Section 3.2.8) and PACE/Titan (Section 3.2.7). Integration of our stochastic scheduler component with these systems will be investigated.

8.2. Scheduler Testing

The testing and validation of the design and principles behind the probabilistic scheduler will be done throughout simulations and a possible deployment of the SO-GRM Grid testbed. Instrumental in this effort is as large as possible collection of accounting and monitoring data from production environments

serving large communities of users. UCL's CCC installation will be the main source of such data, although every effort will be made to obtain similar data from other commercial and academic Grid installations.

In the off-line analysis phase, hypotheses will be formed based on the subsets of data and tested on the complete dataset. Second phase testing will be undertaken when appropriate statistical models have been selected to model user application behaviour and prediction algorithms have been developed. Using simulation tools, a range of jobs with varying adherence to the statistical model will be submitted through the simulated scheduler to establish the adaptability of the prediction engine and its functional envelope. As a result, further fine tuning may be necessary and some empirical observations of the prediction confidence intervals may emerge. Finally, once the first implementation has been done, the GridLoader tool will be used to simulate real Grid applications with probabilistic execution times and resource utilisation values drawn from pre-defined probability distribution functions. Monitoring framework presented in Section 6 will be used to measure the effects of scheduler system overheads, timings of data acquisition and overall system dynamics. The performance of the scheduling logic will be most evident in theoretical simulations run on the production system job traces, while any implementation issues and their effect on the scheduler will show in the tests using the GridLoader application simulator.

8.3. Open Issues

Several issues regarding the implementation, restrictions and intended uses of the probabilistic scheduler remain open.

8.3.1. Unique Grid Process Identification

One of the most valuable meta-data information not available to the Grid schedulers at the moment is the unique identity of the executing process. Many Grid applications are a complex set-up of various data staging and preparation steps, distributed processes and complex result handling mechanisms. These compound jobs are often referred to as Grid workflows, and their management has become a major problem for the Grid community. Current practices rely on the use of scripting languages, and often one generic script can run any number of different applications with widely different characteristics. For these reasons, our stochastic scheduler is not able to fully identify the executable being run, and develop a model for its behaviour. The issue of workflow management and job identification is actively researched by the Grid community, and is likely to be resolved in the near future. Next versions of the Grid middleware should be able to uniquely identify different workflows and their constituent components, and make this information available through an open interface.

8.3.2. Hardware Heterogeneity

The execution time and resource utilisation statistics observed by the proposed probabilistic scheduler are only applicable to a certain node hardware configuration. Although this seems to conflict with a vision of a widely heterogeneous Grid infrastructure, it is in the line with the intended mode of use of such local-level scheduler.

For reasons of administrative, economical and political nature, processing farms constituting a global Grid are highly homogeneous. Choosing one, or at most two, architectures (manufacturer) reduces administrative overheads, creates economies of scale and a more manageable environment. In that context, our scheduler will be able to generate predictions for its local hardware environment, offering those predictions, together with an associated confidence level, as a bid value to the global Grid meta-scheduler. Lacking any previous experience with the offered job, our stochastic scheduler will revert to a batch mode and reflect such uncertainty in the confidence level offered. It is on the Grid meta-scheduler to pool bids from different clusters and select the most appropriate one, based on its own set of requirements and restrictions.

Such hardware-specific approach has been adopted after considerable research into profiling and predicting application performance on different hardware platforms. As previously discussed in the Section 3.1.2, execution predictions, of acceptable quality, reached using moderate system resources, in or near real-time, on a widely varying architectures as found on the Grid, and with many application's performance sensitive to very specific hardware capabilities simply may not be possible.

8.3.3. Data Storage and Communication

Adequate means of compressing, storing and communicating statistics pertaining to a large number of jobs, metrics and meta-factors will be required for efficient operation of the scheduler. Any compression method used should endeavour to maintain the statistical properties of the compressed data, hence wavelets and other similar methods will be investigated.

Underlying monitoring and accounting sub-systems will handle communications and storage aspects of the raw data, but the scheduler will be required to communicate its own statistical models and predictions both within the local cluster and to the higher level meta-scheduler. Ideas from previous work within the SO-GRM project group will be used as the starting point for those scheduler aspects. Of special interest are XML encoded antibodies used in the I^3 component, and self-organising gossip-like communication protocol employed in SORD component.

8.4. Business Plan Development

Grid computing has drawn interest from a large number of commercial institutions, and from large organisations from across a broad range of industries. Common to all is a need for large pool of computational power, cost of which is now more determined by the level of utilisation then by the price of the installed equipment.

Effective scheduling plays an important part in keeping the hardware loaded, and providing high levels of return on investment made into the data-centres. It is the author's belief that proposed probabilistic scheduling holds a very attractive value promise and that an operational system could be successfully marketed to large commercial Grid operators.

Subject to further market research, a proposal for the commercialisation of this novel scheduling technology would be made in the form of a business plan. Funding will also be sought for further studies of system feasibility and possible revenue stream. Throughout this process, adequate steps will be taken in cooperation with UCL's technology transfer office to affirm any intellectual property rights that may be applicable to the system.

9. CHAPTER NINE

APPENDICES

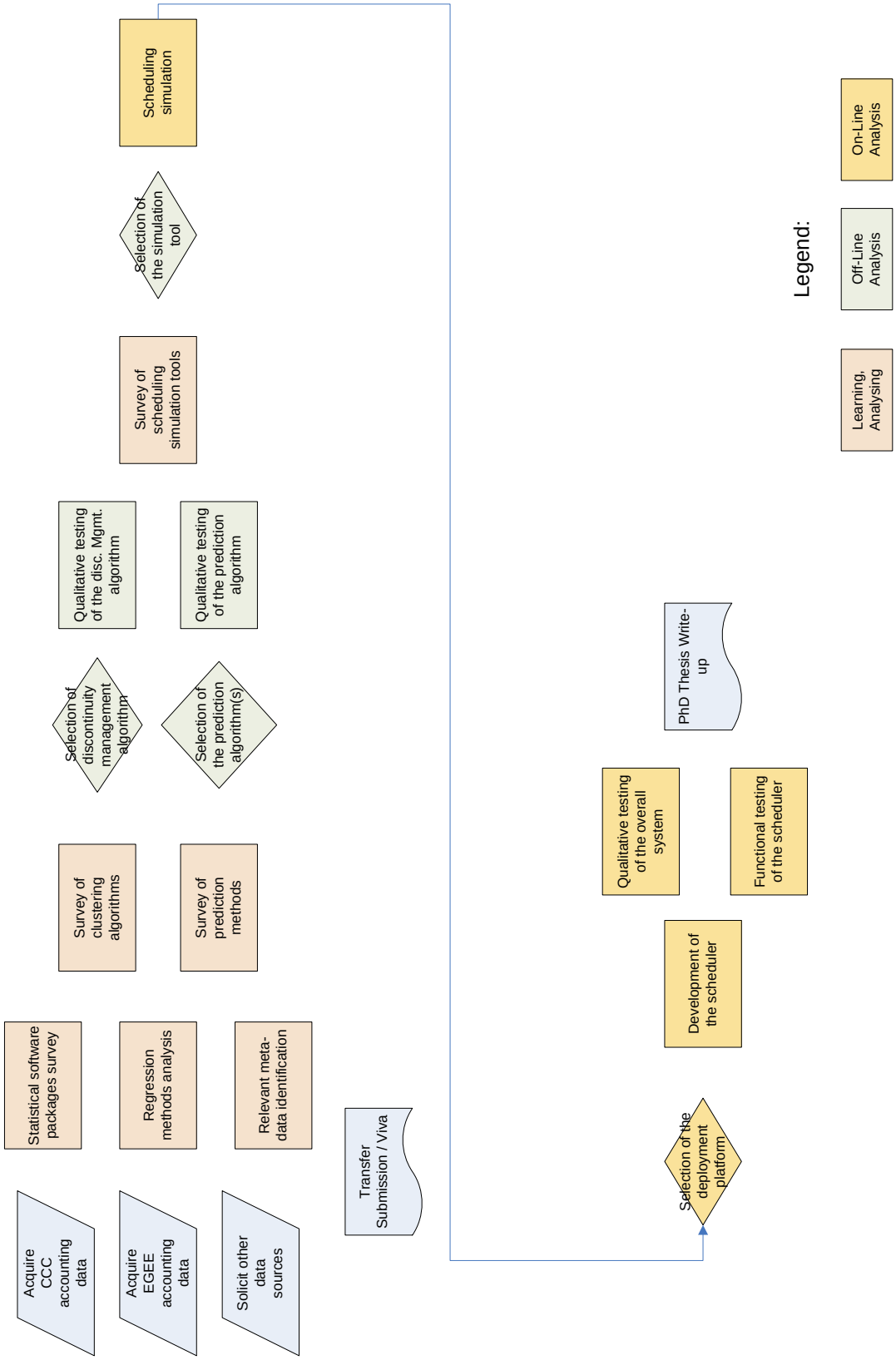
9.1. Glossary of Terms

Acronym	Meaning
AppLeS	Application Level Scheduling
ASCI	Accelerated Strategic Computing Initiative
CCC	UCL Central Computing Cluster
CF	RRD Database Consolidation Function
CPU	Central Processing Unit
DEC	Digital Equipment Corporation (now part of HP)
DS	RRD Database Data Source
FIFO	First In First Out
FLOPS	Floating Point Instructions Per Second
FRFO	First Ready First Out
GASS	Globus Access to Secondary Storage
GGF	Global Grid Forum
GIIS	Grid Information Index Service
GIS	Globus Information Service
GMA	Grid Monitoring Architecture
GRAM	Globus Resource Allocation Manager
GRIS	Grid Resource Information Service
GSI	Globus Security Infrastructure
IP	Internet Protocol
LDAP	Lightweight Directory Access Protocol
LSF	Load Sharing Facility
MDS	Globus Monitoring & Discovery Service
MIPS	Millions of Instructions Per Second

Acronym	Meaning
MPI	Message Passing Interface
NWS	Network Weather Service
OGSA	Open Grid Services Architecture
PBS	Portable Batch System
PDF	Probability Distribution Function
PE	GridSim Processing Elements
PID	Process Identifier
PKI	Private Key Infrastructure
RDBMS	Relational Database Management System
R-GMA	Relational Grid Monitoring Architecture
RRA	Round Robin Archive
RRD	Round Robin Database
SGE	Sun Grid Engine
SLA	Service Level Agreement
SLAM	SO-GRM SLA Management Component
SMP	Symmetric Multiprocessor
SOAP	Simple Object Access Protocol
SORD	Self-Organised Resource Discovery Protocol
SQL	Simple Query Language
SSH	Secure Shell
Tcl	Tool Command Language
TCP	Transport Control Protocol
TLS	Transport Layer Security
ToS	Type of Service
UDP	User Datagram Protocol
URI	Universal Resource Identifier
VO	Virtual Organisation
WSRF	Web Services Resource Framework
XDR	External Data Representation
XML	eXtensible Mark-up Language

9.2. Table of Figures

9.3. Work Plan


Scheduling simulation
Selection of the simulation tool
Survey of scheduling simulation tools
Qualitative testing of the disc. Mgmt. algorithm
Qualitative testing of the prediction algorithm
Selection of discontinuity management algorithm
Selection of the prediction algorithm(s)
Survey of clustering algorithms
Survey of prediction methods
Statistical software packages survey
Regression methods analysis
Relevant meta-data identification
Acquire CCC accounting data
Acquire EGEE accounting data
Solicit other data sources
Transfer Submission / Viva
Selection of the deployment platform
Development of the scheduler
Qualitative testing of the overall system
Functional testing of the scheduler
PhD Thesis Write-up
Legend:
On-Line Analysis
Off-Line Analysis
Learning, Analysing

9.4. Publications and Relevant Documents

9.4.1. London Communications Symposium 2003

"Resource and Application Models for Advanced Grid Schedulers"

Aleksandar Lazarevic, Lionel Sacks

ABSTRACT: As Grid computing is becoming an inevitable future, managing, scheduling and monitoring dynamic, heterogeneous resources will present new challenges. Solutions will have to be agile and adaptive, support self-organization and autonomous management, while maintaining optimal resource utilisation. Presented in this paper are basic principles and architectural concepts for efficient resource allocation in heterogeneous Grid environment.

Available at: www.ee.ucl.ac.uk/~alazarev/papers/

9.4.2. International Symposium on Telecommunications

"Self-organising management of Grid environments"

Ioannis Liabotis, Ognjen Prnjat, Tope Olukemi, Adrian Li Mow Ching, Aleksandar Lazarevic, Lionel Sacks, Mike Fisher, Paul McKee

ABSTRACT: This paper presents basic concepts, architectural principles and algorithms for efficient resource and security management in cluster computing environments and the Grid. The work presented in this paper is funded by BTExacT and the EPSRC project SO-GRM (GR/S21939).

Available at: www.ee.ucl.ac.uk/~alazarev/papers/

9.4.3. Next Generation Networking - Multi-Services Networks

"Adaptive Grid Scheduling and Resource Management"

Aleksandar Lazarevic, Lionel Sacks

Available at: www.ee.ucl.ac.uk/~alazarev/papers/

9.4.4. London Communications Symposium 2004

"Measuring and Monitoring Grid Resource Utilisation"

Aleksandar Lazarevic, Lionel Sacks

ABSTRACT: Effective resource utilisation monitoring and highly granular yet adaptive measurements are prerequisites for a more efficient Grid scheduler. We present a suite of measurement applications able to monitor per-process resource utilisation, and a customisable tool for emulating observed utilisation models.

Available at: www.ee.ucl.ac.uk/~alazarev/papers/

9.4.5. The Ninth IFIP/IEEE International Symposium on Integrated Network Management

"Enabling Adaptive Grid Scheduling and Resource Management"

Aleksandar Lazarevic, Lionel Sacks, Ognjen Prnjat

ABSTRACT: Wider adoption of the Grid concept has led to an increasing amount of federated computational, storage and visualisation resources being available to scientists and researchers. Distributed and heterogeneous nature of these resources renders most of the legacy cluster monitoring and management approaches inappropriate, and poses new challenges in workflow scheduling on such systems. Effective resource utilisation monitoring and highly granular yet adaptive measurements are prerequisites for a more efficient Grid scheduler. We present a suite of measurement applications able to monitor per-process resource utilisation, and a customisable tool for emulating observed utilisation models. We also outline our future work on a predictive and probabilistic Grid scheduler. The research is undertaken as part of UK e-Science EPSRC sponsored project SO-GRM (Self-Organising Grid Resource Management) in cooperation with BT.

Available at: www.ee.ucl.ac.uk/~alazarev/papers/

9.5. Code Listing

9.5.1. GridLoader

```
/*
Grid Loading Application
Aleksandar Lazarevic,
Dept of E&E Engineering, University College London
v 0.8

*/


// -------- Include Files --------

#include <stdio.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <signal.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

// -------------------------------



// -------- Static Definitions ---

//Used for LoadIdle() sleep time, in useconds
#define CPU_IDLE_PAR_A 150000

//Used for LoadIdle() sleep time, in useconds
#define CPU_IDLE_PAR_B 8

// CPU Load Pareto A parameter in useconds
#define CPU_LOAD_PAR_A 1000

// 1: Print out a lot of debuging info, 0: silent operation
#define DEBUG 0

// -------------------------------

// -------- Functions ------------

void LoadNet(float run_for, int burst_delay, char *to_addr);
void LoadMem(int mem_amount);
int LoadCPU(float run_for, float cpu_load_par_b);
void LoadIdle();
void Done();
void LoadCPUDone();
void LoadNetDone();
float ran_pareto(float A, float B);

// -------------------------------


// -------- Global Variables -----

typedef enum {DONE = 0, NET_LOAD, MEM_LOAD, CPU_LOAD, IDLE} State_Type;

State_Type curr_state;    // States init table
```

```
int LoadNetAlarm = 0;    // Signal handler for Net loop, break on =1
int LoadCPUAlarm = 0;    // Signal handler for CPU loop, break on =1
double LoadCPUTime = 0;                          // Keeps track of CPU time already
done, in seconds

// ------------------------------



// -------- main () --------------

main (int argc, char *argv[])
{


    if (argc != 7)
    {

        printf("Grid Loading Application, v.0.8.\n\nERROR: Must have exactly 6 runtime
parameters.\n\nGridload utilisation:\n\tgridload <NET> <CPU> <MEM> <BURST> <IP>
<PARETO_B>\n\nWhere:\n\t<NET> = NET transfer time in seconds - FLOATING POINT\n\t<CPU>
= Total CPU loading time in seconds - FLOATING POINT\n\t<MEM> = Amount of memory to
allocate in MB - INTREGER\n\t<BURST> = Interpacket sleep time in useconds -
INTREGER\n\t<IP> = IP address to send network traffic to - DOTTED
NUMERICAL\n\t<PARETO_B> - LoadCPU time Pareto B parameter - FLOATING POINT\n\nAs no
rigorous error cheking is done, please abide by the specification of these
paramters.\n");
        exit(1);
    }



    int i = 0;
    time_t *mytime;
    long int t;

    //Allocate enough memory for type time_t
    mytime = malloc(sizeof(time_t));

    //Get current Epoch time
    t = time(mytime);

    if (DEBUG)
        printf("Seeding with Epoch time: %d\n", t);

    //Seed the random genereator with current Epoch, has a resolution of a second! May
cause problems later
    srand48(t);



    float net_run_for, cpu_run_for, cpu_load_par_b;
    int mem_amount, burst_delay;


    // Assign parameters to variables

    net_run_for = atof(argv[1]);
    cpu_run_for = atof(argv[2]);
    mem_amount = atoi(argv[3]);
    burst_delay = atoi(argv[4]);
    cpu_load_par_b = atof(argv[6]);

    if (DEBUG)
        printf("Arguments: %f,%f,%d,%d,%s,%f\n", net_run_for, cpu_run_for, mem_amount,
burst_delay, argv[5], cpu_load_par_b);


    curr_state = NET_LOAD;   // Jump to first state

    while (curr_state)
    {
        switch (curr_state)
        {

        case 1 :
```

```
            LoadNet ( net_run_for, burst_delay, argv[5] );
            break;

        case 2 :
            LoadMem ( mem_amount );
            break;

        case 3 :
            LoadCPU ( cpu_run_for, cpu_load_par_b);
            break;

        case 4 :
            LoadIdle();
            break;
        }
    }
}

// ------------------

void LoadNet (float run_for, int burst_delay, char *to_addr)
{

    if (DEBUG)
        printf("Entering LoadNet()\n");

    int udp_socket, i;
    char send_this[1400]; // MTU max 1500, use 1400+headers+spare
    struct itimerval NetTimer;
    struct sockaddr_in IP_client;
    long run_for_sec, run_for_usec;

    // Convert float run_for into two long /sec and /usec
    run_for_sec = floor (run_for);
    run_for_usec = 1000000 * (run_for - run_for_sec);

    //Signal handler for CLOCK_REAL
    signal (14, *LoadNetDone);

    //Fill message with letters A
    for (i = 0; i < 1400; i++)
        send_this[i] = 'A';

    //Set timer
    NetTimer.it_interval.tv_sec = 0;
    NetTimer.it_interval.tv_usec = 1;
    NetTimer.it_value.tv_sec = run_for_sec;
    NetTimer.it_value.tv_usec = run_for_usec;

    //Open socket
    udp_socket = socket (PF_INET, SOCK_DGRAM, 0);

    IP_client.sin_family = AF_INET;
    IP_client.sin_addr.s_addr = inet_addr (to_addr);
    IP_client.sin_port = 22222;

    if (DEBUG)
        printf("Entering loop, timer set: seconds:%d , useconds:%d\n",
NetTimer.it_value.tv_sec, NetTimer.it_value.tv_usec);

    //Run the timer
    setitimer (ITIMER_REAL, &NetTimer, 0);

    while (!LoadNetAlarm)
    {
        sendto (udp_socket, &send_this, sizeof(send_this), 0, (struct sockaddr *)
&IP_client, sizeof(IP_client));
        usleep(burst_delay);
    }

    curr_state = MEM_LOAD;

    //Reset Timer Raiser back to 0 !!!
    LoadNetAlarm = 0;

    if (DEBUG)
```

```
        printf("Leaving LoadNET()\n");

    // return(); if needs to return something to main()

}

void LoadNetDone()
{
    LoadNetAlarm = 1;
}


// ---------------------------------



// ---------------------------------

int LoadCPU (float run_for, float cpu_load_par_b)
{

    if (DEBUG)
        printf("Entering LoadCPU()\n");

    //If done enough time return setting state DONE
    if (LoadCPUTime >= run_for)
    {
        curr_state = DONE;

        if (DEBUG)
            printf("Leaving LoadCPU - time required has been reached!\n");

        return (0);
    }

    float this_run;
    struct itimerval CPUTimer;
    long this_run_usec;

    //Signal handler for CLOCK_VIRTUAL
    signal (26, *LoadCPUDone);

    //Choose how long to run this time around
    this_run = ran_pareto ( CPU_LOAD_PAR_A, cpu_load_par_b );


    this_run_usec = floor (this_run);

    //Set the timer
    CPUTimer.it_interval.tv_sec = 0;
    CPUTimer.it_interval.tv_usec = 0;
    CPUTimer.it_value.tv_sec = 0;
    CPUTimer.it_value.tv_usec = this_run_usec;

    if (DEBUG)
        printf("Setting timer: seconds:%d , useconds:%d\n", CPUTimer.it_value.tv_sec,
CPUTimer.it_value.tv_usec);

    //Run the timer
    setitimer (ITIMER_VIRTUAL, &CPUTimer, 0);

    //Load the CPU using an empty loop - very effective :)
    while (!LoadCPUAlarm)
        ;

    //Update how much time we spent in the CPUload loop, converting timer useconds
into seconds
    LoadCPUTime += (double)this_run_usec / 1000000;

    if (DEBUG)
        printf("LoadCPUTime so far: %.12f\n", LoadCPUTime);


    curr_state = IDLE;

    //Reset Timer raiser back to 0 !!!
```

```
    LoadCPUAlarm = 0;

    if (DEBUG)
        printf("Leaving LoadCPU - this_run done\n");

}

void LoadCPUDone()
{
    LoadCPUAlarm = 1;
}
// -------------------------------------


// -------------------------------------

float ran_pareto (float A, float B)
{

    double x, y1, y2, y;

    //Get a random value between (0.0,1.0]
    x = drand48();

    if (DEBUG)
        printf("Random number is: %f\n", x);


    //Use inverse method to obtain a random Pareto number
    y1 = 1 - x;
    y2 = - ( (1 / B) * ( log(y1) ) );
    y = A * ( exp(y2) );


    if (DEBUG)
        printf("Returning Pareto number:%f\n", y);

    //Return floating random Pareto
    return (y);

}

// -------------------------------------

// -------------------------------------

void LoadMem (int mem_amount)
{

    if (DEBUG)
        printf("Entering LoadMEM()\n");

    char *data;
    int j;

    //Allocate the required memory as given by mem_amount. SIZEOF(CHAR) = 1
    data = (char *) malloc ( 1024 * 1024 * mem_amount * sizeof(char));

    //Above line increases virtual memory size, loop below makes kernel allocate
physical memory
    for (j = 0;j < 1024*1024*mem_amount;j++)
        data[j] = 'A';

    //All done here, memory will be free when process terminates
    curr_state = CPU_LOAD;

    if (DEBUG)
        printf("Leaving LoadMEM(), allocated: %d MB\n", mem_amount);
}

// -------------------------------------

// -------------------------------------

void LoadIdle ()
{
```

```
    if (DEBUG)
        printf("Entering LoadIdle()\n");

    long int sleep_for;

    //Get a Pareto using predefined A and B, and round it down
    sleep_for = floor ( ran_pareto (CPU_IDLE_PAR_A, CPU_IDLE_PAR_B));

    if (DEBUG)
        printf("Will be sleeping for:%d  useconds\n", sleep_for);


    //Sleep for so many useconds
    usleep(sleep_for);

    // Include sleeping time in CPUTime - more realistic, try to agree with 'time
./gridload ...'
    LoadCPUTime += (double)sleep_for / 1000000;

    //Return back to LoadCPU(), which controls when to break out of LoadCPU<->LoadIdle
loop
    curr_state = CPU_LOAD;

    if (DEBUG)
        printf("Leaving LoadIdle()\n");

}

// -----------------------------------
```

9.5.2. Matlab® Parameter File Generator

```
% Aleksandar Lazarevic - v0.8
% Create RUN file for Globus Grid Loader
% Simple format for testing without SORD & I3


% Output format:
% <NET> <CPU> <MEM> <BURST> <IP> <PARETO_B> <NEXTREQ DELAY> <NEXTREQ IP>


clear;

% Change the values below ...

IP_LOW = 146;
IP_HIGH = 149;
IP_PREFIX = '128.16.235.';

CPU_TOTAL_PARETO_A = 1500;
CPU_TOTAL_PARETO_B = 6;

CPU_LOAD_PARETO_B_MIN = 2;
CPU_LOAD_PARETO_B_MAX = 10;

MEM_MEAN = 40;
MEM_MIN = 180;

NET_MEAN = 15;
NET_MIN = 20;

BURST_MEAN = 8;
BURST_MIN = 5;

NEXTREQ_MIN = 2000;
NEXTREQ_MAX = 3000;

NEXT_HOST_MIN = 13;
NEXT_HOST_MAX = 13;
NEXT_HOST_PREFIX = 'android-ee';
```

```
ITERATIONS = 200;

% No need to change anything below this line!
% ----------------------------------------


% Put Pareto numbers in CPU

x=rand(ITERATIONS,1);
for i=1:ITERATIONS,
   y1=1-x(i);
   y2=-(1/CPU_TOTAL_PARETO_B)*(log(y1));
   CPU(i)= CPU_TOTAL_PARETO_A * exp(y2);
end
% ---------------------------

% Generate random net transfer times
NET = NET_MIN + NET_MEAN * abs ( randn(1,ITERATIONS) );
% ---------------------------


% Generate random rounded memory sizes for MEMLOAD
MEM = round ( MEM_MIN + MEM_MEAN * abs ( randn(1,ITERATIONS) ) );
% ---------------------------

% Generate random rounded burst delay intervals for NETLOAD
BURST = round ( BURST_MIN + BURST_MEAN * abs ( randn(1,ITERATIONS) ) );
% ---------------------------

% Generate random IP address for NETLOAD traffic and next request
IPLoad = round (IP_LOW + (IP_HIGH - IP_LOW) * rand(1,ITERATIONS) );

IPNext = round (IP_LOW + (IP_HIGH - IP_LOW) * rand(1,ITERATIONS) );
% ---------------------------


% Generate random next request delay
NEXTREQ = round (NEXTREQ_MIN + (NEXTREQ_MAX - NEXTREQ_MIN) * rand(1,ITERATIONS) );
% ---------------------------

% Generate random CPU_LOAD_PARETO_B values from range MIN to MAX
CPU_LOAD_PARETO_B = CPU_LOAD_PARETO_B_MIN + (CPU_LOAD_PARETO_B_MAX -
CPU_LOAD_PARETO_B_MIN) * rand(1,ITERATIONS);
% ---------------------------

% Generate random address for NEXT_HOST
NEXTHOST = round (NEXT_HOST_MIN + (NEXT_HOST_MAX - NEXT_HOST_MIN) * rand(1,ITERATIONS)
);
% ---------------------------

% File writing

RUN_FILE = fopen('param.grid', 'w');
STAT_FILE = fopen('stats.csv','w');

for i=1:ITERATIONS

    fprintf(RUN_FILE, '%f %f %d %d %s%d %f %10s%d %d\n', NET(i), CPU(i), MEM(i),
BURST(i), IP_PREFIX, IPLoad(i), CPU_LOAD_PARETO_B(i), NEXT_HOST_PREFIX, NEXTHOST(i),
NEXTREQ(i) );
    fprintf(STAT_FILE, '%f,%f,%d\n', NET(i), CPU(i), MEM(i) );

end

fclose(RUN_FILE);
fclose(STAT_FILE);
% --------------------



% Plotting
hold on;

subplot(3,3,1);
plot (CPU,'k');
axis auto;
```

```
title('CPU Load');
ylabel('Seconds');

subplot(3,3,2);
hist(CPU,100);
axis auto;
title('CPU Load Histogram');
xlabel('Seconds');
ylabel('Count');

subplot(3,3,4);
plot (MEM,'r');
axis auto;
ylabel('MBytes');
title('Memory Allocation');

subplot(3,3,3);
axis auto;
plot (NET,'m.');
title('NET Transfer Time');
ylabel('Seconds');

subplot(3,3,5);
hist(MEM,100);
axis auto;
title('Memory Histrogram');
xlabel('MBytes');
ylabel('Count');

subplot(3,3,6);
axis auto;
plot (BURST,'b.');
ylabel('uSeconds');
title('Packet Burst Delay')


subplot(3,3,7);
hist(NEXTREQ,100);
axis auto;
title('NEXTREQ Time');
xlabel('Seconds');
ylabel('Count');

subplot(3,3,8);
hist(CPU_LOAD_PARETO_B,100);
axis auto;
title('CPU Load B Time');
xlabel('Value');
ylabel('Count');


% --------------------------------
```

9.5.3. Ganglia Custom Metric Broadcast Script

```
#!/bin/sh

#
# 'broadcast_metric' Script
#
# Generate Ganglia XML and publish certain local system metric
# to Ganglia MetaDeamon through custom information provider
# 'gmetric'
#
# Uses 'ps' to obtain info and 'gawk' to process it
# then sleeps for a period of time
#
#
# (c) Aleksandar Lazarevic 2004

# Metric name to be published
METRIC="globus-cpu-percentage"
```

```
# 'ps' query command (part of)
# use "-U username" or "-C cmdline" or "-p PID"
CMD="-C gl08"

# Polling period, don't make it shorter then 5s due to CPU util.
SLEEP=5

# ------------------------------


while [ 0 ]
do

VAL=`ps ${CMD} -o pcpu | gawk '{s += $1} END {print s}'`

gmetric --name="${METRIC}" --value=${VAL} --type=float

sleep ${SLEEP}

done
```

9.5.4. Round-Robin Database Data Sweep Script

```
#!/bin/sh

# 'sweeprrd' Script
#
# Extract highest frequency data from Ganglia RRD for a given metric
# Checks last timestamp in local file and exports from the next
# timestamp up to NOW less 45 seconds (don't ask why!)
#
# It then sleeps for some time. Polling should be less then 60 minutes
# otherwise high freq data will be lost or consolidated (RRD)
#
#
# (c) Aleksandar Lazarevic 2004
#
#

# RRD File location WITH TRAILING SLASH!!!
RRDLOC="/var/lib/ganglia/rrds/UCL/android-ee13.cs.ucl.ac.uk/"

# RRD metric database to use
RRDFILE1="globus-cpu-percentage.rrd"
RRDFILE2="cpu_idle.rrd"

# Raw data file (will contain timestamp:value[scientific]
FILE1="/home/aleks/experiments/4/globus_load"
FILE2="/home/aleks/experiments/4/total_load"

# Sleep time (less than 3500 more than 120)
SLEEP=900

# Debug?
DEBUG=0

# ------------------------------


# export $FILE1 $FILE2

while [ 0 ]
do

STARTTIME=`tail -n 1 $FILE1 | gawk '/:/ {print substr($1,1,10)}'`


if [ "$STARTTIME" == "" ]
then
        STARTTIME=`date +%s`
```

```
        let "STARTTIME-=3500"

        if [ $DEBUG = 1 ]; then
        echo No previous startime found. Using: $STARTTIME
        fi

else
        let "STARTTIME+=15"

        if [ $DEBUG = 1 ]; then
        echo Previous startime: $STARTTIME
        fi
fi

rrdtool fetch ${RRDLOC}/${RRDFILE1} AVERAGE -r 1 --end=now-45 --start=$STARTTIME |
grep : >> $FILE1
rrdtool fetch ${RRDLOC}/${RRDFILE2} AVERAGE -r 1 --end=now-45 --start=$STARTTIME |
grep : >> $FILE2

sleep ${SLEEP}

done
```

9.5.5. Simple Batch Scheduler (Globus flavour)

```
#!/bin/sh

#
# 'scheduler' Script
#
# Simple master-slave schedulng script
# Take a runfile with GridLoader parameters and two extra: next host and delay
# Using globus-job-run to submit to "next host" after "delay"
#
# MUST BE RUN AS SOGRM!
# [Because of Globus certificates]
#
#
# (C) Aleksandar Lazarevic 2004
#
#

# Debug?
DEBUG=0

# Runfile Location
RUNFILE="/home/aleks/experiments/4/param"

# Runtime Log
RUNLOG="/home/aleks/experiments/4/run.log"

# Globus certificate password
GLOBUSPASS="/home/aleks/scripts/globus-proxy-pass"
# -----------------------------------



# Open files
exec 3<$RUNFILE
exec 4>>$RUNLOG

# count how many jobs to run (-1)
jobs=`wc -l ${RUNFILE} | gawk '{print $1}'`

if [ $DEBUG = 1 ]
then echo Jobs to run:$jobs
fi


# Main Loop
```

```
for ((i=1; a < $jobs ; a++))
do
  read -a V <&3

  # Add NET time and CPU time - NO Floating Point ???
  #$(( runtime = V[0] + V[1] ))

  if [ $DEBUG = 1 ]
      then echo Total Runtime:${runtime} Next Host: ${V[6]}  in-: ${V[7]}
  fi

  if [ $DEBUG = 0 ]
      then
        echo -n `date +%s`, >&4
        grid-proxy-init -q -valid 24:00 -pwstdin <$GLOBUSPASS
        globus-job-run ${V[6]} /home/aleks/gridloader/gl08 ${V[0]} ${V[1]} ${V[2]}
${V[3]} ${V[4]} ${V[5]}
        grid-proxy-destroy
        echo `date +%s`,${V[6]} >&4
  fi

if [ $DEBUG = 1 ]
then

    echo sent it! now go to sleep
    sleep 1

else

        sleep 45
    # sleep ${V[7]}

fi

done
# End of main loop
exit 0
```

9.5.6. Simple Batch Scheduler (SSH flavour)

```
#!/bin/sh

#
# 'scheduler' Script
#
# Simple master-slave schedulng script
# Take a runfile with GridLoader parameters and two extra: next host and delay
# Using SSH submit to "next host" after "delay"
#
# MUST BE RUN AS ROOT!
# [Because of SSH certificates]
#
#
# (C) Aleksandar Lazarevic 2004
#
#

# Debug?
DEBUG=0

# Runfile Location
RUNFILE="/home/aleks/experiments/3/param"

# Runtime Log
RUNLOG="/home/aleks/experiments/3/run.log"

# -----------------------------------



# Open files
exec 3<$RUNFILE
```

```
exec 4>>$RUNLOG

# count how many jobs to run (-1)
jobs=`wc -l ${RUNFILE} | gawk '{print $1}'`

if [ $DEBUG = 1 ]
then echo Jobs to run:$jobs
fi


# Main Loop

for ((i=1; a < $jobs ; a++))
do
  read -a V <&3

  # Add NET time and CPU time - NO Floating Point ???
  #$(( runtime = V[0] + V[1] ))

  if [ $DEBUG = 1 ]
      then echo Total Runtime:${runtime} Next Host: ${V[6]}  in-: ${V[7]}
  fi

  if [ $DEBUG = 0 ]
      then
        echo -n `date +%s`, >&4
        ssh ${V[6]} /home/aleks/gridloader/gl08 ${V[0]} ${V[1]} ${V[2]} ${V[3]} ${V[4]}
${V[5]}
        echo `date +%s`,${V[6]} >&4
  fi

if [ $DEBUG = 1 ]
then

    echo sent it! now go to sleep
    sleep 1

else

        sleep 45
    # sleep ${V[7]}

fi

done

# End of main loop

exit 0
```

9.6. References

[1] "LCG Computing Fabric Overview," http://lcg-computing-fabric.web.cern.ch/LCG-Computing-Fabric/fabric_presentations/lhcc_review_fabric_overview_BPanzer3.ppt; Last Accessed: 2003

[2] "SETI@home: Search for Extraterrestrial Intelligence at home," http://setiathome.ssl.berkeley.edu/; Last Accessed: 2005

[3] "Folding@Home Distributed Computing," http://www.stanford.edu/group/pandegroup/folding/; Last Accessed: 2005

[4] Foster, Kesselman and Tuecke "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications* vol. 15(3), 2001.

[5] I. Foster, C. Kesselman, J. Nick and S. Tuecke "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," *Global Grid Forum* 2002.

[6] "The Globus Alliance," http://www.globus.org/; Last Accessed: 2005

[7] "EGEE - Gateway," http://egee-intranet.web.cern.ch/egee-intranet/gateway.html; Last Accessed: 2005

[8] "Global Grid Forum," http://www.ggf.org/; Last Accessed: 2005

[9] "Platform Computing - Accelerating Intelligence - Grid Computing," http://www.platform.com/; Last Accessed: 2005

[10] "Avaki : Home," http://www.avaki.com/; Last Accessed: 2005

[11] "United Devices, Inc. ™ - Grid Computing Solutions - Home," http://www.ud.com/home.htm; Last Accessed: 2005

[12] "10 Emerging Technologies That Will Change Your World", 2004;

[13] K. Vairavan and R.A. DeMillo "On the Computational Complexity of a Generalised Scheduling Problem," *IEEE Trans. Computing* vol. C-25, no. 11, pp. 1067-1073, 1976.

[14] M.J. Gonzales "Deterministic Process Scheduling," *ACM Computer Surveys* vol. 9, no. 3, pp. 173-204, 1997.

[15] E.S. Buffa "Modern Production Management", 5th Edition, Wiley; New York, NY, 1977;

[16] R.W. Conway, W.L. Maxwell and L.W. Miller "Theory of Scheduling", Addison-Wesley; Reading, MA, 1967;

[17] M.J. Flynn "Very High-speed Computing Systems," *Proceeding of the IEEE* vol. 54, pp. 1901-1909, 1966.

[18] Casavant and Kuhl "A taxonomy of scheduling in general-purpose distributed computing systems," *Software Engineering, IEEE Transactions on* vol. 14, no. 2 SN - 0098-5589, pp. 141-154, 1988.

[19] S.A. Jarvis, D. Spooner, H. Keung and G. Nudd "Performance prediction and its use in parallel and distributed computing systems," *Parallel and Distributed Processing Symposium, 2003. Proceedings. International* pp. 8 pp. 2003.

[20] Y. Lingyun, I. Foster and J.M. Schopf "Homeostatic and tendency-based CPU load predictions," *Parallel and Distributed Processing Symposium, 2003. Proceedings. International* no. SN - 1530-2075, pp. 9 pp. 2003.

[21] J.M. Schopf and F. Berman "Using stochastic intervals to predict application behavior on contended resources," *Parallel Architectures, Algorithms, and Networks, 1999. (I-SPAN '99) Proceedings. Fourth InternationalSymposium on* pp. 344-349, 1999.

[22] L. Byoung Dai and J.M. Schopf "Run-time prediction of parallel applications on shared environments," *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on* no. SN -, pp. 487-491, 2003.

[23] S.A. Jarvis, D. Spooner, H. Keung, J. Dyson, Z. Lei and G. Nudd "Performance-based middleware services for grid computing," *Autonomic Computing Workshop, 2003* pp. 151-159, 2003.

[24] D.P. Spooner, S. Jarvis, J. Cao, S. Saini and G. Nudd "Local grid scheduling techniques using performance prediction," *Computers and Digital Techniques, IEE Proceedings*- vol. 150, no. 2, pp. 87-96, 2003.

[25] Berman, Wolski, Casanova, Cirne, Dail, Faerman, Figueira, Hayes, Obertelli, Schopf, Shao, Smallen, Spring, Su and Zagorodnov "Adaptive computing on the Grid using AppLeS," *Parallel and Distributed Systems, IEEE Transactions on* vol. 14, no. 4 SN - 1045-9219, pp. 369-382, 2003.

[26] "Condor-G," http://www.cs.wisc.edu/condor/condorg/; Last Accessed: 2005

[27] J. Frey, T. Tannenbaum, I. Foster and S. Tuecke "Condor-G: a computation management agent for multi-institutional grids," *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on* no. SN -, pp. 55-63, 2001.

[28] "Grid Engine Project Home Page," http://gridengine.sunsource.net/; Last Accessed: 2005

[29] "N1 Grid Engine 6," http://www.sun.com/software/gridware/index.xml; Last Accessed: 2005

[30] D. Abramson, R. Sosic, J. Giddy and B. Hall "Nimrod: a tool for performing parametrised simulations using distributed workstations," *High Performance Distributed Computing, 1995., Proceedings of the Fourth IEEE International Symposium on* pp. 112-121, 1995.

[31] "Nimrod/G," http://www.csse.monash.edu.au/~davida/nimrod/nimrodg.htm; Last Accessed: 2005

[32] R. Buyya, D. Abramson and J. Giddy "Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid," *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on* vol. 1, pp. 283-289 vol.1, 2000.

[33] "PBS Pro Home," http://www.pbspro.com/; Last Accessed: 2005

[34] "Beowulf.org: The Beowulf Cluster Site," http://www.beowulf.org/; Last Accessed: 2005

[35] "OpenPBS," http://www.openpbs.org/; Last Accessed: 2005

[36] R.L. Henderson "Job Scheduling Under the Portable Batch System," *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing* pp. 279-294, 1995.

[37] "Platform Computing - Products - Platform LSF," http://www.platform.com/products/LSF/; Last Accessed: 2005

[38] S. Zhou "LSF: Load Sharing in Large-scale Heterogeneous Distributed Systems," *Workshop on Cluster Computing* 1992.

[39] "University of Warwick: Computer Science: Research: PACE," http://www.dcs.warwick.ac.uk/research/hpsg/pace/pace-introduction.html; Last Accessed: 2005

[40] "LeSC - London e-Science Centre - ICENI," http://www.lesc.ic.ac.uk/iceni/; Last Accessed: 2005

[41] L. Young, S. McGough, S. Newhouse and J. Darlington "Scheduling Architecture and Algorithms within the ICENI Grid Middleware," 2003.

[42] "Maui Cluster Scheduler," http://www.clusterresources.com/products/maui/; Last Accessed: 2005

[43] D. Jackson, Q. Snell and M. Clement "Core Algorithms of the Maui Scheduler," *Lecture Notes in Computer Science* vol. 2221, pp. 87-??, 2001.

[44] "Legion: A Worldwide Virtual Computer," http://legion.virginia.edu/index.html; Last Accessed: 2005

[45] "NetSolve," http://icl.cs.utk.edu/netsolve/; Last Accessed: 2005

[46] "Ninf Project Home Page," http://ninf.apgrid.org/; Last Accessed: 2005

[47] P. Sang-Min and K. Jai-Hoon "Chameleon: a resource scheduler in a data grid environment," *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on* no. SN -, pp. 258-265, 2003.

[48] "MGRID - MGRID Accounting," http://www.mgrid.umich.edu/projects/mars.html; Last Accessed: 2005

[49] "SimGrid," http://juggler.ucsd.edu/simgrid/; Last Accessed: 2005

[50] Casanova "Simgrid: a toolkit for the simulation of application scheduling," *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on* no. SN -, pp. 430-437, 2001.

[51] Legrand, Marchal and Casanova "Scheduling distributed applications: the SimGrid simulation framework," *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on* no. SN -, pp. 138-145, 2003.

[52] "Micro Grid," http://www-csag.ucsd.edu/projects/grid/microgrid.html; Last Accessed: 2005

[53] "Grid Application Development Sofware Project (GrADS)," http://www.hipersoft.rice.edu/grads/; Last Accessed: 2005

[54] X. Huaxia, Dail, Casanova and Chien "The MicroGrid: using online simulation to predict application performance in diverse grid network environments," *Challenges of Large Applications in Distributed Environments, 2004. CLADE 2004. Proceedings of the Second International Workshop on* no. SN -, pp. 52-61, 2004.

[55] H.J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura and A.A. Chien "The MicroGrid: a Scientific Tool for Modeling Computational Grids," *Supercomputing* 2000.

[56] "GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing," http://www.buyya.com/gridsim/; Last Accessed: 2005

[57] "SimJava," http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/; Last Accessed: 2005

[58] R. Buyya and M. Murshed "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing", John Wiley & Sons, Ltd.; 2003;

[59] "Ganglia," http://ganglia.sourceforge.net/; Last Accessed: 2005

[60] "RRD TOOL -- About RRDtool," http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/; Last Accessed: 2005

[61] "Ganglia Cluster Toolkit:: Rocks Network Grid Report," http://meta.rocksclusters.org/Rocks-Network/; Last Accessed: 2005

[62] "R-GMA:Relational Grid Monitoring Architecture.," http://www.r-gma.org/; Last Accessed: 2005

[63] B. Tierney "A Grid Monitoring Architecture," 2005.

[64] "NWS," http://www.nsf-middleware.org/documentation/NMI-R3/0/NWS/; Last Accessed: 2005

[65] Wolski "Forecasting network performance to support dynamic scheduling using the network weather service," *High Performance Distributed Computing, 1997. Proceedings. The Sixth IEEE International Symposium on* no. SN -, pp. 316-325, 1997.

[66] Takefusa, Matsuoka, Nakada, Aida and Nagashima "Overview of a performance evaluation system for global computing scheduling algorithms," *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on* no. SN -, pp. 97-104, 1999.

[67] R. Wolski, N.T. Spring and J. Hayes "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Gener. Comput. Syst.* vol. 15, no. 5-6, pp. 757-768, 1999.

[68] R. Wolski "Dynamically forecasting network performance using the Network Weather Service," *Cluster Computing* vol. 1, no. 1, pp. 119-132, 1998.

[69] "GridMon - Grid Network Performance Monitoring for UK e-Science," http://gridmon.dl.ac.uk/; Last Accessed: 2005

[70] "Hawkeye," http://www.cs.wisc.edu/condor/hawkeye/; Last Accessed: 2005

[71] Z. Xuechai, J.L. Freschl and J.M. Schopf "A performance study of monitoring and information services for distributed systems," *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on* no. SN - 1082-8907, pp. 270-281, 2003.

[72] "SO-GRM - EPSRC Web Site - Grants on the Web," http://gow.epsrc.ac.uk/ViewGrant.ASPx?Grant=GR/S21939/01&bannerlink=Programme%20support; Last Accessed: 2005

[73] A. Li Mow Ching, Sacks and McKee "SLA Management and Resource Monitoring for Grid Computing," *London Communications Symposium* 2003.

[74] I. Liabotis, O. Prnjat, T. Olukemi, A.L. Ching, A. Lazarevic, L. Sacks, M. Fisher and P. McKee "Self-organising management of Grid environments," *International Symposium on Telecommunications* 2003.

[75] D.J. Watts "Small Worlds", Princeton University Press; 1999; ISBN:0691005419

[76] I. Liabotis, O. Prnjat and L. Sacks "Policy-Based Resource Management for Application Level Active Networks," 2001.

[77] Prnjat, Liabotis, Olukemi, Sacks, Fisher, McKee, Carlberg and Martinez "Policy-based management for ALAN-enabled networks," *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on* no. SN -, pp. 181-192, 2002.

[78] R.O. Duda, P.E. Hart and D.G. Stork "Pattern Classification", 2nd Edition, John Wiley and Sons; 2001;

[79] O. Prnjat, T. Olukemi, I. Liabotis and L. Sacks "Integrity and Security of the Application Level Active Networks," *IFIP Workshop on IP and ATM Traffic Management* 2001.

[80] "Open Source Native XML Database," http://exist.sourceforge.net/; Last Accessed: 2005

[81] Citron "MisSPECulation: partial and misleading use of spec CPU2000 in computer architecture conferences," *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on* no. SN - 1063-6897, pp. 52-59, 2003.

[82] Amato and Dale "Probabilistic roadmap methods are embarrassingly parallel," *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on* vol. 1, no. SN -, pp. 688-694 vol.1, 1999.

[83] "GLUE Schema," http://www.globus.org/mds/glueschemalink.html; Last Accessed: 2005

[84] C. Bishop "Neural Networks for Pattern Recognition", Oxford University Press; Oxford, England, 1995;