Developing a Trusted Computational Grid

John Brennan

School of

Computing and Engineering

University of Huddersfield

A thesis submitted for the degree of

Master of Science

April 2014

Copyright Statement

Abstract

Within institutional computing infrastructure, currently available grid middlewares are considered to be overly complex. This is largely due to behaviours required for untrusted networks. These behaviours however are an integral part of grid systems and cannot be removed. Within this work the development of a grid middleware suitable for unifying institutional resources is proposed. The proposed system should be capable of interfacing with all Linux based systems within the QueensGate Grid (QGG) campus grid, automatically determining the best resource for a given job. This allocation should be done without requiring any additional user effort, or impacting established user workflows. The framework was developed to tackle this problem. It was simulated, utilising real usage data, in order to assess suitability for deployment. The results gained from simulation were encouraging. There is a close match between real usage data and data generated through simulation. Furthermore the proposed framework will enable better utilisation of campus grid resources, will not require modification of user workflows, and will maintain the security and integrity of user accounts.

Acknowledgements

I would like to express my gratitude to my supervisor Dr Violeta Holmes for the useful comments, remarks and engagement through the learning process of this masters thesis.
I would like to thank all members of the HPC-RG:- Ibad Kureshi, Stephen Bonner, Matthew Newall, Shuo Liang and Yvonne James for always being there when I needed to talk through ideas or just rant about my day.
I must give special thanks to Ibad Kureshi for introducing me to the topic, supporting me along the way and collaborating on the code for the cluster simulator. Special thanks are also given to Stephen Bonner for his collaboration on the Java based Hadoop code.
Finally I would like to thank my parents and my partner for their inexhaustible emotional support.

Contents

Document Word Count 17058

List of Figures

List of Tables

List of Abbreviations

AD Active Directory

API Application Programming Interface

AWS Amazon Web Services

BDII Berkeley Database Information Index

BOINC Berkeley Open Infrastructure for Network Computing

CA Certificate Authority

CCLRC Council for the Central Laboratory of the Research Councils

CE Compute Element

CentOS Community ENTerprise Operating System

CPU Central Processing Unit

DHTC Distributed High Throughput Computing

DN Distinguished Name

DQS Distributed Queuing System

DSA Digital Signature Algorithm

EGI European Grid Infrastructure

EMI European Middleware Initiative

EMIv3 European Middleware Initiative V3 [Monte Bianco]

FTP File Transfer Protocol

GFS Google File System

GID Group IDentity

GIIS Grid Index Information Service

GLUE Grid Laboratory Uniform Environment

GOCDB Grid Operations Centre DataBase

GPU Graphical Processing Unit

GRD Global Resource Director

GSI Grid Security Infrastructure

GT Globus Toolkit

GUI Graphical User Interface

HDFS Hadoop Distributed File System

HPC High Performance Computing

HTC High Throughput Computing

HTCondor High Throughput Condor

IaaS Infrastructure as a Service

JDL Job Description Language

JISC Joint Information Systems Committee

JVM Java Virtual Machine

KVM Kernel-based Virtual Machine

LCMAPS Local Credential Mapping System

LDAP Lightweight Directory Access Protocol

LIM Load Information Manager

LSLIB Load Sharing Library

LSF Load Sharing Facility

MDS Metacomputing Directory Service

MPI Message Passing Interface

NASA National Aeronautics and Space Administration

NCAS National Center for Supercomputing Applications

NES National e-Infrastructure Service [Formally National Grid Service (NGS)]

NGS National Grid Service

NSF National Science Foundation

OGSA Open Grid Service Architecture

OGE Oracle Grid Engine

OS Operating System

OSCAR Open Source Cluster Application Resources

OSG Open Science Grid

PaaS Platform as a Service

PBS Portable Batch System

PCC Platform Computing Corporation

PKI Public Key Infrastructure

POVB Pool Of Virtual Boxes

PSK Pre-Shared Key

PXE Pre-Boot Execution Environment

QGG QueensGate Grid

QoS Quality of Service

RA Registration Authority

RB Resource Broker

RDO Red Hat Distribution of OpenStack

RES Remote Execution Server

RHEL Red Hat® Enterprise Linux®

RPM Red Hat Package Manager

RSA Rivest-Shamir-Adleman

SaaS Software as a Service

SGE Sun Grid Engine

SLA Service Level Agreement

SSH Secure SHell

TFTP Trivial File Transfer Protocol

UEC Ubuntu Enterprise Cloud

UEE Universal Execution Environment

UI User Interface

UID User IDentity

UoH University of Huddersfield

UoM University of Manchester

VDT Virtual Data Toolkit

VM Virtual Machine

VO Virtual Organisation

VOMS Virtual Organisation Management System

WMS Workload Management System

XSEDE Extreme Science and Engineering Discovery Environment

Chapter 1

Introduction

1.1 An overview of 'The Grid'

Grid computing was defined in the 1990s as computing infrastructure which could be configured in such a way to allow large-scale resource sharing (Magoulès, 2010). Within computational science, grids play an important role in making disparate resources accessible to end users. A great deal of work has been done in this field producing a number of grid middlewares. In this project, the currently available grid middlewares will be considered. Primarily, grid middlewares integrate with underlying cluster middlewares and schedulers, so it was important to consider these also. There is increasing pressure for grid solutions to be able to interface with a wider range of end resource types. Cloud computing is becoming an attractive option for allowing system scaling to be performed easily. Traditionally, grid middlewares have generally been considered to be rather complex by users and administrators alike. This is largely due to the security measures required when sharing resources across global networks. Consequently, classic grid middlewares are not looked upon favourably when considering unification of resources within an institution.

1.2 The University of Huddersfield QueensGate Grid

At the University of Huddersfield (UoH) the High Performance Computing (HPC) and High Throughput Computing (HTC) resources fall under the umbrella of the QGG campus grid. The QGG is made up of a number of distinct resources. While there is a common connection point for most systems, upon reaching this common point users would branch out to their preferred system. Some systems are entirely independent and must be accessed directly. This multiple system configuration has serious drawbacks. System load balancing is almost impossible to achieve, as this would require users to investigate the state of each suitable resource before submitting a job to any one of them. In some cases, the user may not even know that a resource is suitable and therefore it could be completely overlooked. In addition, more training is required to enable users to use differing systems. Dedicating administrator and user time for training on a single scheduler or batch system can be difficult, even more so in a situation where 4 or 5 different systems may be in use. A novel solution would be required to overcome this situation, present minimal impact on users and improve utilisation of existing resources.

1.3 Aims and Objectives

The overall goal of this project was to produce a method for allowing better unification of campus computing resources. The key objectives for success were:

- Unify submission for as many computing resources as possible.
- Minimise any possible impact to end users.
- Minimise requirement of additional administration time.
- Maintain integrity of user accounting.

Deliverables for this project were considered to be a working proof of concept deployment which would adhere to the aforementioned guidelines.

1.4 Methodology

In order to achieve the research aims and objectives a survey of current grid middleware was conducted. UoH campus grid systems and workflows were investigated, with focus on grid security and deployed job schedulers. A simulator was designed and implemented, which was verified against existing real usage data. A novel job submission framework was designed, implemented and evaluated to enable better utilisation of existing systems within the QGG, and reduce impact on user workflows. This thesis presents the body of work, which has been carried out for this research study, and is organised as follows:

- Chapter 2 - Investigate existing Grid middlewares, batch systems and cloud technologies, also considering existing deployments.
- Chapter 3 - Analyse UoH resources and current architecture.
- Chapter 4 - Investigate Grid security.
- Chapter 5 - Propose and design a solution for easier access and better utilisation of available resources.
- Chapter 6 - Propose a suitable method for evaluation of developed solution.
- Chapter 7 - Implement and evaluate proposed solution(s).

Chapter 2

Literature Review

2.1 Definition of a Grid

As stated in the Introduction grid computing was defined during the mid 1990s as computing infrastructure which could be configured in such a way to allow large-scale resource sharing (Magoulès, 2010). This definition was further clarified by Ian Foster in 1998 (Marowka, 2002), when he stated that a grid should adhere to three fundamental rules:

- Provide coordination for resources which would not otherwise be centrally controlled.
- Use standardised and preferably open-source interfaces and protocols, which should remain as general purpose as possible.
- Provide a significant quality of service.

  (Foster & Kesselman, 1999)

From here it was recognised that such definitions needed to be part of a standardised framework and, as such, the Open Grid Service Architecture (OGSA) was

proposed in the paper 'The Physiology of the Grid, An Open Grid Services Architecture for Distributed Systems Integration' by Ian Foster, et al in 2002 (Foster, Kesselman, Nick, & Tuecke, 2002). The OGSA, Version 1 was published in January 2005 (Foster et al., 2005) and has since become the standard to which almost all grid oriented software development adheres. Open standards are a very high priority in the OGSA in order to allow many different systems to be interoperable. However with all the additional complexities/customisations of different hardware and software stacks maintaining open standards is not always possible. Fundamentally the purpose of a grid is to allow disparate resources, such as CPU/hours, Memory/core and storage space, to be utilised by a pool of users. These users, who are potentially separated by thousands of miles, can then use these resources in order to solve a common problem. The aim of grid infrastructure is to supply greater resources than may be available locally and to allow work to be done much faster. The grid middleware allows jobs to be passed to a local batch system or scheduler which is managing an appropriate geographically distributed resource.

2.2 Grid Middleware Currently in Use

A vast amount of work has been done around the world to produce software that can realise the goals set out by the OGSA. In this section we will look at some of the most commonly used middlewares available today.

2.2.1 The Globus Toolkit

The Globus Toolkit (GT) has become the de-facto standard in grid middleware. It is widely used across across Northern America and to a lesser extent across Europe. Generally speaking most, if not all, other grid middlewares are capable of using the Globus gatekeeper. The GT also provides a simple Certificate Au-

thority (CA) which helps system administrators lay the foundations for grid based authentication. These components, along with the rest of the toolkit, ensure that the GT can enable systems to easily join larger grids so long as certificates are mutually trusted. The major drawback of the GT is that it lacks any kind of resource discovery which can be accessed by the end user from their submit node. Instead GT Metacomputing Directory Service (MDS) can be configured to publish aggregate data to websites which, if made available, can be accessed by the user. This also means that the GT itself cannot perform any kind of load balancing and all decisions on resources where a job will run are left to the user. All the components which make up the GT, including the more recent Web Service components are shown in Figure 2.1.

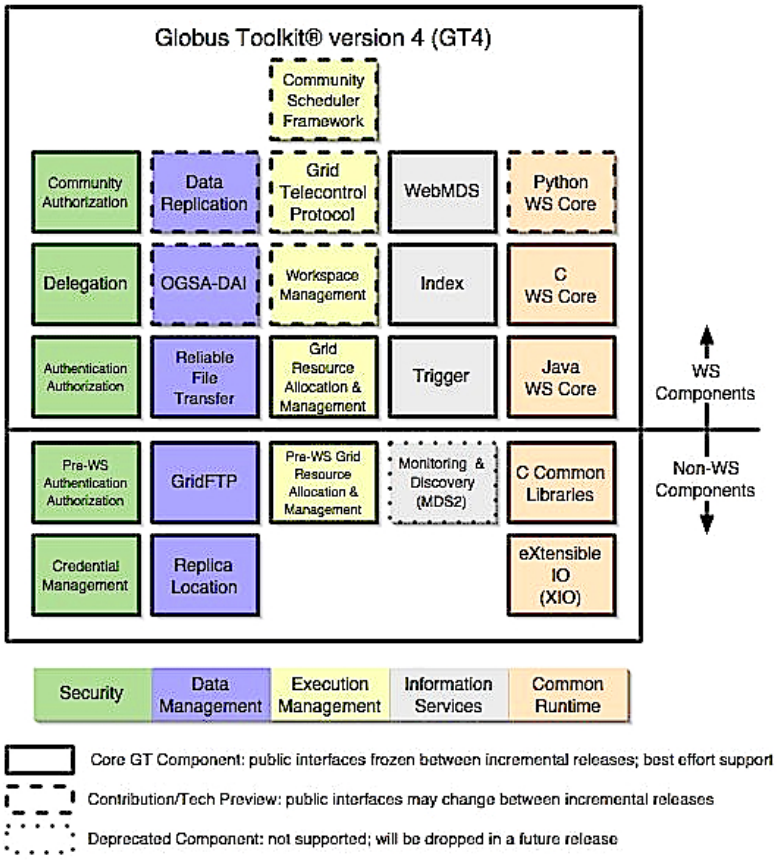

Figure 2.1: GT Architecture. (Boverhof, 2005)

2.2.2 EMI 3

The current European Middleware Initiative (EMI) release is European Middleware Initiative V3 [Monte Bianco] (EMIv3). This is the third and final version of grid software to be released under the EMI. Within this middleware architecture, highlighted in Figure 2.2, users log into a User Interface (UI) from which all resources in their Virtual Organisation (VO) can be discovered through the use of a Berkeley Database Information Index (BDII) based system, and following the Grid Laboratory Uniform Environment (GLUE) version 2 schema. Jobs are submitted to the Workload Management System (WMS) which decides which Compute Element (CE) is best suited for the job, based on user requirements and CE loading among other factors. EMI was the middleware of choice for the UK based National e-Infrastructure Service [Formally National Grid Service (NGS)] (NES). Both of these projects are now at the end of their respective funding cycles. The National Grid Service (NGS) has been almost fully dismantled, with just a few services remaining which are maintained and funded through other sources. The EMI will maintain full support until 30/04/2014, at which time, support for individual components will revert back to individual software developers.

The EMI WMS, which was used as part of the UK national e-infrastructure, was essentially the WMS that was developed under the gLite project. gLite was brought in as part of this larger European project. (Brennan et al., 2013) This middleware is a very functional software which can cope with a great many resources, automating their discovery and dealing with load balancing very well. However it does in its own right require a large amount of resources. In order to have a functional EMI WMS a minimum of five servers are required. While this is acceptable in a national level service, such high resource demands can be difficult to cope with in an institutional setting. One other problem with this system is that, even with a GT installation High Throughput Condor (HTCondor) (as HTCondor was only supported in gLite

3.1) systems seem unable to get the results of jobs back to the WMS. Also, it is unable to support any kind of cloud platform.

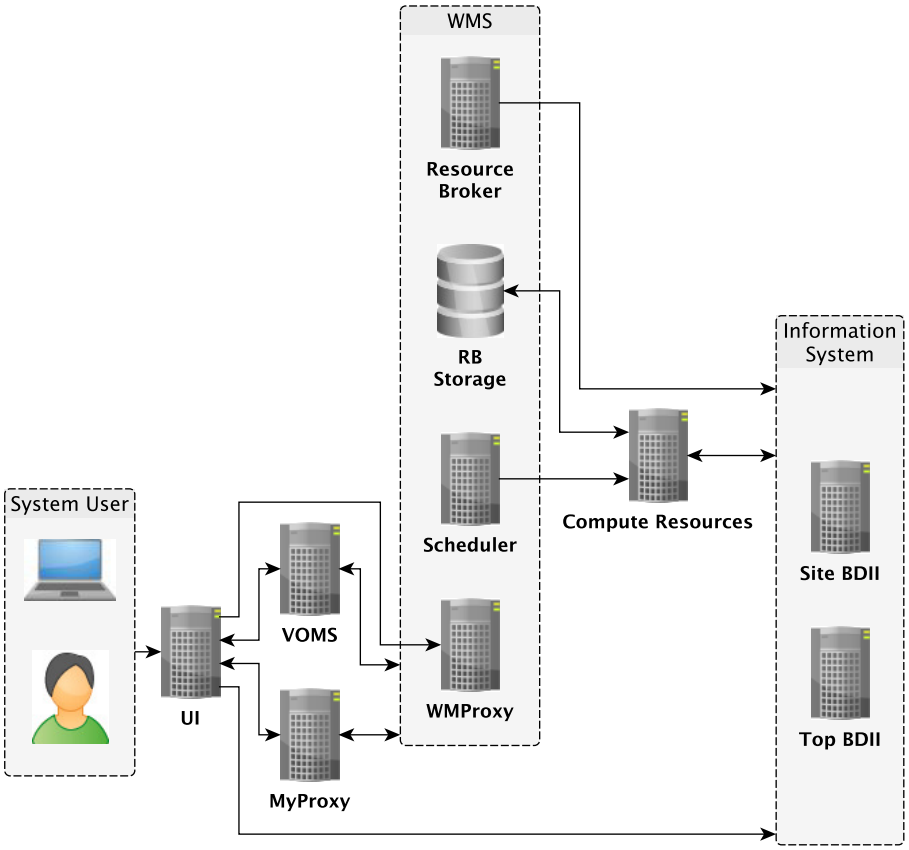

Figure 2.2: UI/WMS Architecture

2.2.3 OSG

The Open Science Grid (OSG) middleware is widely adopted across the USA, and directly integrates with TerraGrid. The OSG project continued the work started by the Virtual Data Toolkit (VDT) project created at The University of Wisconsin-Madison, which is no longer in active development. This project is similar to the EMI. It is fundamentally a collection of software brought together under a common banner to better enable national sharing of resources because everyone is using the same software stack. The software used in the OSG middleware is centred

around the GT, but with explicit support for plugins such as Local Credential Mapping System (LCMAPS) to enable VO support. It gives a standardised framework for all software, right down to the compute node level. As this system is essentially based upon the GT model, use at a command-line level still suffers from the same resource discovery problems that are present in the donor middleware. However, under the OSG project a web based UI called Bosco has been developed explicitly for Distributed High Throughput Computing (DHTC) and makes resource discovery much more convenient. With Bosco, users can log into a web interface which allows them to submit jobs to local resources, including Portable Batch System (PBS) and HTCondor based systems, all OSG resources including cloud technology based systems and any other resources that are explicitly linked with the institution that the user is connecting from. The OSG middleware has not been widely adopted across Europe as the EMI project was already in place and the default pre configuration of the software is very biased towards USA based systems. However, considering the EMI project is coming to an end, the extended functionality of the OSG system does seem to indicate that it could be a viable alternative

2.3 Batch Systems Currently in Use

There are a wide range of batch systems and schedulers in use within the resources available to researchers across the globe. Some batch systems, such as HTCondor, were developed to make better use of existing resources but most are targeted at making the best use of resources on dedicated machines. In the following section some of the most popular solutions will be considered.

2.3.1 PBS, OpenPBS, PBSPro and Torque

The PBS resource manager currently exists in two different 'flavours'. When the PBS project initially started in the early 1990's it was developed by Veridian Information Solutions and funded by National Aeronautics and Space Administration (NASA). Not long after the initial development, Veridian released a commercial version of the software which was called PBSPro. It then followed that the open source version came to be called OpenPBS. Within three years PBSPro was acquired by Altair Engineering who discontinued development of OpenPBS. At this point, development of the open source product was taken over by Adaptive Computing who, having no rights to the PBS brand, changed the name of the software to Torque. Since that point the status-quo has been maintained with the paid for commercial version remaining PBSPro and the open source version being Torque.(Samuel, 2008)

PBS consists of three distinct parts, pbs_server, pbs_sched and pbs_mom. There is one pbs_mom for each compute node within a system, all of which communicate back to a single pbs_server instance. The pbs_server makes a queue for requested jobs, delegates jobs to the pbs_moms and collects all job related information such as time taken etc. The bundled scheduler, pbs_sched, is an interchangeable part of the system which can be replaced with alternatives such as Maui, Moab or even

a custom scheduler. The purpose of the scheduler is to decide which jobs should run next, depending on which resources are available. While PBSPro is capable of running under Microsoft Windows® as well as with *NIX based systems, this capability has not been extended to Torque, which means that Torque is unsuitable for any software not built against a *NIX based system. (Beer, 2008)

2.3.2 Oracle Grid Engine

Sun Grid Engine (SGE) started its life codenamed Codine. It was originally developed in the early 1990's by Fritz Ferstl and was based on Florida State University's Distributed Queuing System (DQS), and eventually evolved into Global Resource Director (GRD). During August 2000 GRD was acquired by Sun Microsystems who renamed the software SGE and, by 2001, they had released versions for Solaris and Linux along with making the source code available Making the source code avaliable helped fuel adoption of the software. SGE has recently undergone another change of name to Oracle Grid Engine (OGE) and the software is currently available in both open source and commercial form.(Univa Corporation, 2013)

The OGE software functionality is provided through three system daemons. These are known as sge_master, sge_schedd and sge_execd. The sge_master controls cluster management and scheduling, sge_qmaster maintains information about queues, hosts, jobs, user permissions, and current load. sge_schedd is the scheduling daemon and maintains an up-to-date view of the cluster status The scheduling daemon makes the decisions pertaining to which jobs are dispatched to which queues, how to reorder and change the priority of jobs to maintain the best use of the system. These decisions are then passed to the sge_master for actioning. The execution daemon, sge_execd is responsible for the queue on the compute node upon which it is running, the execution of the jobs in that queue and the reporting of job statistics and host load, back to the sge_master.(Oracle

Corporation, 2013)

2.3.3 LSF

Load Sharing Facility (LSF) is a system that is developed and distributed by Platform Computing Corporation (PCC). It is a general purpose software, targeted at distributed computing systems. LSF can be considered in terms of separately licensed components which make up the suite. These components are comprised of LSF Multicluster, LSF Batch and LSF JobScheduler all of which run upon the LSF Base system.

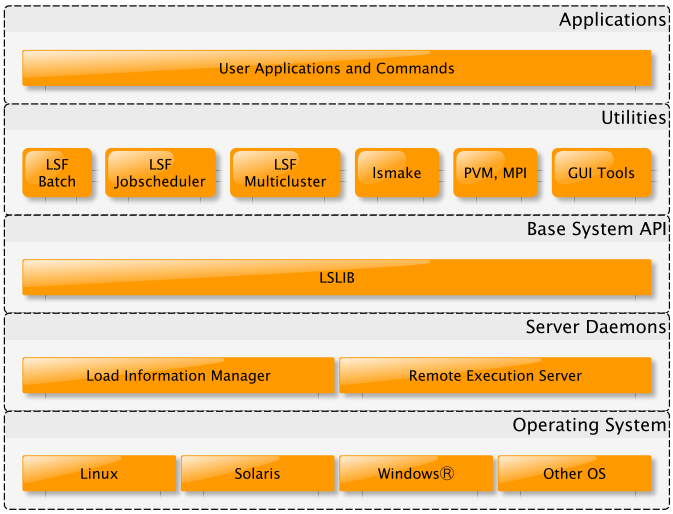

Figure 2.3: LSF Architecture.

Figure 2.3 shows the architecture of LSF within a homogenous system. The base system is composed of three components. These are: the Load Sharing Library (LSLIB), Load Information Manager (LIM) and Remote Execution Server (RES). The LSLIB provides a standard Application Programming Interface (API) for all LSF components to communicate with the LIM and RES, which are both daemons running on the underlying operating systems. All of which allows the user to be presented with a consistent platform independent environment. This

middleware is able to run on a very broad range of different operating systems, making it very versatile, however this is provided through a paid for model only (Platform Computing Corporation, 1996).

2.3.4 HTCondor

HTCondor is a middleware which seeks to utilise CPU cycles which would otherwise be 'wasted' within an environment where there are many standard desktop computers. Many large institutions provide a large number of workstations to be used for general day to day tasks, The problem is that for a significant amount of time these machines are left idle, leaving an expensive resource going to waste. HTCondor runs a service/daemon on Microsoft Windows® or *NIX based systems which advertises itself as a worker node to a management server. This management server, or head node, accepts jobs from a user and then forwards it on to a suitable idle worker node for execution, employing underused resources to solve computational problems. One potential drawback with this system is that when a user wishes to directly use a machine which is being utilised by the system, HTCondor must immediately stop any processing so there is no impact on the user. This can cause a loss of data as, in most cases, any work already done is 'dumped' and the job is passed to another worker node which will start the process from the beginning, making the software unsuitable for long running jobs. The team developing the middleware developed a solution which can mitigate this problem somewhat. Where there is source code available for the software being used, HTCondor has the ability to compile an executable with specific 'hooks' allowing checkpointing, meaning that the worker node can periodically report a jobs progress along with any data to a checkpointing server. If a worker node then has to 'dump' that job then the next worker node can get the progress information from the checkpoint server and continue the job from the point that the last checkpoint

save was made, leading to far fewer lost cycles and much better overall utilisation of available systems. Another limitation of the HTCondor middleware is that although it is capable of running parallel jobs which require a Message Passing Interface (MPI) environment, to do this requires that a HTCondor pool has specially configured dedicated machines and cannot be achieved within the standard cycle stealing model. (Thain, Tannenbaum, & Livny, 2005)

2.3.4.1 POVB

Within the HTCondor ecosystem there is a general shortcoming. This is because most institutional general purpose computers with Microsoft Windows® based operating systems installed, whereas a large degree of computational science requires *NIX type systems. This drawback is addressed by the Pool Of Virtual Boxes (POVB) project. POVB can be installed upon a Microsoft Windows® based host. During this process a Linux based VirtualBox Virtual Machine (VM) is created which has HTCondor installed. The created virtual machine is then controlled by POVB passing required input to the virtualised HTCondor instance. This configuration creates a Windows® host that advertises itself to the HTCondor head node as a Linux resource and can process any Linux based software. The VM within POVB is always started at system boot time and controlled dynamically. When a user is physically present and using the resource in question then the VM is scaled back, in terms of memory and Central Processing Unit (CPU) usage, as far as is possible to minimise any impact of this system on the user. Once the system has been idle for a predetermined period of time then the VM is 'woken up' and the resources available to it are increased to allow for maximum utilisation of the resource.(Herzfeld, Olson, & Struble, 2010)

2.3.5 Warewulf Cluster Manager

Warewulf is a scalable systems management suite originally developed to manage High Performance Computing (HPC) Linux clusters. It was focused on general systems management with scaleability as a primary concern. The system includes a framework for system management, configuration, pre developed image provisioning or installation, monitoring for filesystem changes, and event notification. Warewulf is developed around a modular architecture allowing behaviour modification through the development of additional modules. This provides a scaleable and customisable cluster provisioning tool which is suitable for a wide range of applications within HPC environments. Warewulf can configure a shared environment within a cluster via an image-like file system on the master node. These images can then be provisioned to designated nodes through Pre-Boot Execution Environment (PXE) utilising Trivial File Transfer Protocol (TFTP) (US Department of Energy, 2013).

2.3.6 OSCAR

Open Source Cluster Application Resources (OSCAR) allows administrators, who may have a limited experience of *NIX type systems, to install a Beowulf or classic type HPC cluster. The OSCAR packages contain many common HPC cluster softwares, for example Torque and C3 tools. These packages are presented through a dedicated management tool allowing simple selection of packages, with a great deal of the configuration being handled automatically. It also provides support for administrators to develop custom packages to produce any required behaviour outside the scope of the standard packages (Kim, 2009).

OSCAR is installed onto a HPC cluster head node on top of a supported Linux distribution. The supported distributions include Red Hat® Enterprise Linux®

(RHEL), Debian and Ubuntu, although only RHEL supports the OSCAR suite in its entirety. During installation the system administrator selects the packages required for installation. OSCAR then installs the required packages on the head node and also creates an installation specific disk image for use on the compute nodes. This image is then pushed to, and installed on, the configured nodes through PXE and TFTP. This provides an easily scaleable and robust suite with very strong management capabilities, which was used to win the first ever Cluster Challenge competition held at Supercomputing 2007 (Greidanus & Klok, 2008).

The default OSCAR setup directly supports scientific computing using a MPI implementation; several MPI packages are included in the default OSCAR packages. OSCAR makes use of the 'modules' command/binary to allow switching between multiple versions of MPI versions either at system or user level (Naughton et al., 2002).

2.4 National Grids: Past and Present

In this section we will look at how grids have been implemented at a national and international level. These grids versatile as, by their very nature, they potentially have extremely large user bases. Challenges like this are difficult to overcome and, while much work has been done to make grids efficient and user friendly, there are still many areas which warrant further work. This is especially true in the area of interoperability to enable the use of as many different software packages as possible.

2.4.1 National e-Infrastructure Service (NES)

The NES was a Joint Information Systems Committee (JISC) and Council for the Central Laboratory of the Research Councils (CCLRC) funded project which ran

between 2003 and 2013 with the mission ”To provide coherent electronic access for UK researchers to all computational and data based resources and facilities required to carry out their research, independent of resource or researcher location” (Richards, 2007). The service was initially composed of compute clusters at Oxford and Leeds Universities, a data cluster at Manchester University and an e-Science funded data centre. To bring this infrastructure together the GT (then in version 2) was used. This facilitated the provision of a UK CA allowing users to register with the service and have secure credentials issued. These credentials could then be used to access the resources provided within the grid. Grid Security Infrastructure (GSI) enabled Secure SHell (SSH). It was available on some systems to allow secure interactive terminal sessions, secure file transfer. The Globus gatekeeper allows for direct submission to service endpoints. By 2007, with phase 2 up and running, the core services provided by the NES had grown significantly into the following list;

- UK CA and Registration Authority (RA) Network
- Helpdesk / User Support
- Documentation
- Training (www.nesc.ac.uk)
- Website(s), Wiki
- MyProxy (National Service + Java Client Upload tools)
- Portal (NGS Application Repository)
- INCA / Grid Integration Test Script Monitoring
- Berkeley Database Information Index
- Storage Resource Broker
- User Accounting System Registration, Accounting, Policing
- Advanced Reservation
- gLite Resource Broker / UI

- Grid Operations Centre DataBase (GOCDB)
- GSISSH Term
- Grid RegIstry with Metadata Oriented Interface: Robustness, Efficiency, Security (Universal Description, Discovery, and Integration)

These services were being utilised by 500 users and were underpinned by 48 compute nodes with Dual Socket Dual Core AMD Opteron 280 processors, 8GB memory, 2x80GB disks, Myrinet 2000; 8 compute nodes with Quad Socket Dual Core AMD Opteron 280 processors, 32GB memory, 2x80GB disks, Myrinet 2000; 8 storage nodes Dual Socket Dual Core AMD Opteron 280 processors, 8GB memory, 2x80GB disks, Myrinet 2000, Fibre HBA. 5 x 12 TB Infotrend Storage Arrays and a Qlogic 5200 SANbox Testing using HPL on 240 Cores gave 934Gflops (81% peak) (Richards, 2007). When funding for the NGS ended in 2013 there were 10 partners and 19 affiliates (NGS, 2013) providing and utilising a wide range of resources which were largely coupled together with the EMI UI/WMS middleware. All of the centrally managed infrastructure, apart from the CA, have now been taken down and resource sharing has to be to be negotiated between individual institutions.

2.4.2 Extreme Science and Engineering Discovery Environment

The Extreme Science and Engineering Discovery Environment (XSEDE) falls into four separate categories. Is can be considered as project, an institution, a set of services and a VO. XSEDE provides widely distributed infrastructure, support services and technical expertise. XSEDE supports advanced computing, high-end visualisation, data analysis, and is funded by the National Science Foundation (NSF). The XSEDE project has a $121 million grant award made by the NSF to the National Center for Supercomputing Applications (NCAS) at the University of Illinois and its partners in order to run for five years. XSEDE is the successor to

the NSF funded TeraGrid project. Institutionally XSEDE is a collaboration of 18 partner organisations working together to deliver a series of services. As a set of services, XSEDE integrates clusters, visualisation, data collections and software into a unified virtual system. XSEDE also provides authentication and security mechanisms allowing access to file systems, remote job submission, monitoring, file transfer services, support services and a user portal (Reilly et al., 2013).

2.4.3 European Data Grid

The main goal of the Data Grid project was to develop an infrastructure that would enable scientific collaboration, bringing together researchers regardless of their geographical location to share data resources on a grand scale. The project developed software solutions and test beds to handle many petabytes of distributed data, hundreds of computing resources and many simultaneous users. The test-bed environment spanned twenty major sites in Europe, the USA, and Asia-Pacific. It was operational throughout the project and offered more than one thousand CPUs and several terabytes of storage. The test bed was used to explore the potential grid technologies, in particular grid schedulers, data and fabric management systems and grid information services. It was tested by applications from multiple scientific disciplines. DataGrid fostered the wide-range use of grid computing by allowing researchers to assess the benefits of grid technology in a large-scale test. Additionally, an extensive tutorial programme targeted end-users, with more than twenty events across the globe, which were attended by around seven hundred people who received training. The resources from these sessions were made available to universities and were used in several university courses. The software and infrastructure developed for Data Grid project was evaluated in high-energy physics, biology and earth sciences. These applications contributed to the development of high-level tools like the portal Genius that made grid environments

easier to use. The DataGrid software is available under an open source, OSI-approved software licence (Gagliard, 2001).

2.4.4 World Community Grid

World Community Grid runs on Berkeley Open Infrastructure for Network Computing (BOINC), developed at University of California, Berkeley, USA with funding from the NSF and sponsorship from IBM. This allows anyone to install a service on their personal computer which will enable their computer, or more specifically idle CPU cycles, available to users that submit jobs to the World Community Grid. This type of crowd-sourced grid has the potential to provide a great deal of computational power for a very low cost. Made possible by using compute time that may otherwise go un-utilised. However, due to the nature of using domestic machines which are not always on, the amount of compute resources can vary greatly. (World Community Grid, 2013)

2.4.5 OurGrid

Cycle stealing Peer to Peer grid, Developed in Java to provide a free to join and free to use distributed grid for individuals. This grid is not tied to any large resources or clusters and has been running since 2004. Again as a cycle stealing grid the potential for compute power is enormous. The same issues are present here as in WGC discussed above. Maintaining suitable levels of available resources can prove challenging. (Mowbray, 2007)

2.5 Campus Grids

In this section we will consider some current campus grids and how the local administrators have overcome challenges to deliver services tailored to their users.

2.5.1 The University of Reading Campus Grid

The University of Reading grid is fundamentally a HTCondor pool which is composed of around five hundred worker nodes. These worker nodes have Microsoft Windows® XP and CoLinux installed, allowing both Windows® and Linux binaries to be run, without the need to have multi-boot systems. As only a single system it is not really a true grid but simply a number of HTCondor pools with campus wide flocking enabled. (University of Reading, 2013)

2.5.2 University of Cambridge Campus Grid

According to Calleja et al. (2008) CamGrid, the computing grid at the University of Cambridge, is now primarily composed of HTCondor pools. There is a large centrally managed pool which is augmented by other flocked pools administered by individual schools within the university, allowing campus wide resource sharing. While CamGrid has no central provision for a PBS based cluster it is potentially supported through allowing HTCondor to submit jobs to PBS. Therby users to run jobs which require tightly coupled MPI networks. There are some issues with this system, one being that the PBS queue must be on the same machine as the condor_schedd. This means that any PBS based resource which is added in this manner will not truly be an integral part of the CamGrid as a whole. The systems implemented in this way will only be accessible from a school controlled HTCondor head-node and not from the grid primary entry point which is controlled centrally (*CamGrid and PBS University Computing Service*, 2013). While HTCondor can

natively run MPI type jobs it is far from ideal as systems in HTCondor pools are not very tightly coupled. Considering this along with the fact that HTCondor requires dedicated resources for MPI universe jobs and cannot flock MPI jobs between pools, it highlights a significant weakness within the deployment. (Thain et al., 2005)

2.5.3 University of Oxford Campus Grid

The University of Oxford recognised almost a decade ago that with individual schools purchasing resources to fulfil their own needs, it was very likely that a situation would arise where some departments may have excess resources, whereas others may struggle to complete their work on what was available to them. To avoid this situation, and to allow researchers access to external resources, the decision was taken to implement campus wide resource sharing through a common middleware. This system was to provide data storage along with computational resources. Built around four main components, an information server, resource broker, VO manager and a data vault, the system became known as OxGrid (Wallom & Trefethen, 2006).

Figure 2.4 shows that OxGrid provides a single 'control system' which is the gatekeeper, that all users of the system, internal or external to the university, must connect through it. This control system is built around a VDT software stack utilising a Condor-G based Resource Broker (RB) and encompassing VO support. Through this the users, with personal X-509 certificates, are able to utilise PBS, SGE and HTCondor through a single submission interface (Wallom, 2007).
In order for this system to work all resource endpoints must advertise their capabilities and supported VOs to an information server. When a user submits a job, the RB must first check a users VO membership with a Virtual Organisation Manage-

Figure 2.4: OxGrid Diagram (Wallom & Trefethen, 2006)

ment System (VOMS) server. Then it checks which resources that particular VO is able to use, at which point a job can actually be submitted to one of the shortlisted endpoints utilising the GT (Wallom, 2007).

2.5.4 University of Manchester Campus Grid

The University of Manchester (UoM) describes clusters interconnected through the use of GT as 'super-clusters' which can then be considered as a single shared resource (University of Manchester, 2013). According to Allan (2011) the UoM maintains a total of 69 nodes with 2.5TB of storage, all made available to researchers through the GT middleware. The GT GSISSH is used as the primary access mechanism for this system, therefore all users must be in possession of a valid X-509 certificate. The underlying middleware is PBS. Although these systems are capable of being shared nationally through the GT it seems that as there is currently no funding model for national sharing, The UoM has decided to only maintain internal access to these systems.

2.6 Distinction Between HPC and HTC

What are the differences between HPC and High Throughput Computing (HTC)? HTC differs from HPC in that it is designed to process tasks that require fairly short processing times, usually minutes or hours and are either purely serial or embarrassingly parallel, and need to be run through hundreds or thousands of iterations. Whereas HPC systems are designed to handle tasks which take large amounts of CPU time and/or require tightly coupled parallel networks.

2.7 Cloud Computing

Cloud computing has evolved from Grid computing and research in the field of HPC (Foster, Zhao, Raicu, & Lu, 2008). Cloud computing can provide Infrastructure as a Service (IaaS), while Grid and HPC generally provide Platform as a Service (PaaS) and Software as a Service (SaaS). This allows users of a Cloud system to access, via a web interface or methods such as SSH, infrastructure which has peen provisioned to their specifications. Cloud systems make excellent use of resources allowing, for example, very high memory instances to be provisioned which only exist for the time they are being used. This means institutions and companies using Cloud technology can very effectively scale resources, without having to constantly fund purchases of hardware which may only be useful for a single task (Chang, Bacigalupo, Wills, & De Roure, 2010). There are a number of commercial Cloud providers, such as Amazon's EC2 and Azure. However these use proprietary software and are obviously pay-per-use which generally puts them out of reach of institutions such as the University of Huddersfield (UoH). These are also often put out of reach for research purposes, due to the pay per use model, as it can be difficult to determine costs at the beginning of a project. A

study performed at the National Technical University of Athens, Greece found it would have been cheaper to create private cloud rather than provision resources on commercial cloud services (Konstantinou, Floros, & Koziris, 2012). To that end, we will only consider open source Cloud middlewares within this section.

2.7.1 OpenStack

OpenStack first appeared in late 2010 and was developed as a joint project between Rackspace Hosting and NASA, with the aim to provide a standardised cloud platform available to all. In late 2012 the OpenStack Foundation was formed and assumed responsibility for the project. Since that time over 200 companies have joined the foundation to help either financially or with various other aspects of the business (The OpenStack Foundation, 2013). OpenStack has a 6 month release cycle and is currently fully supported by Debian, Ubuntu and RedHat. The OpenStack deployment can change quite drastically between releases, making it a potentially problematic software option for system administrators. However going forward, this may become less of an issue, due to enterprise level distributions such as Red Hat Distribution of OpenStack (RDO). OpenStack is released under the Apache Licence and is comprised of nine individual components, some of which are essential for a deployment and some which are not.

Figure 2.5 shows the communication paths present between each of the OpenStack modules. Nova Compute is the module responsible for integrating with an existing system hypervisor, such as Kernel-based Virtual Machine (KVM), to control the actual instances of VMs that are spawned to service user requests.

Swift is the object storage module which offers data replication across all available hardware, ensuring data is always available even in the case of a disk failure.

Cinder block storage allows various forms of storage to be attached to instances within the cloud, allowing users to have immediate direct access to existing data

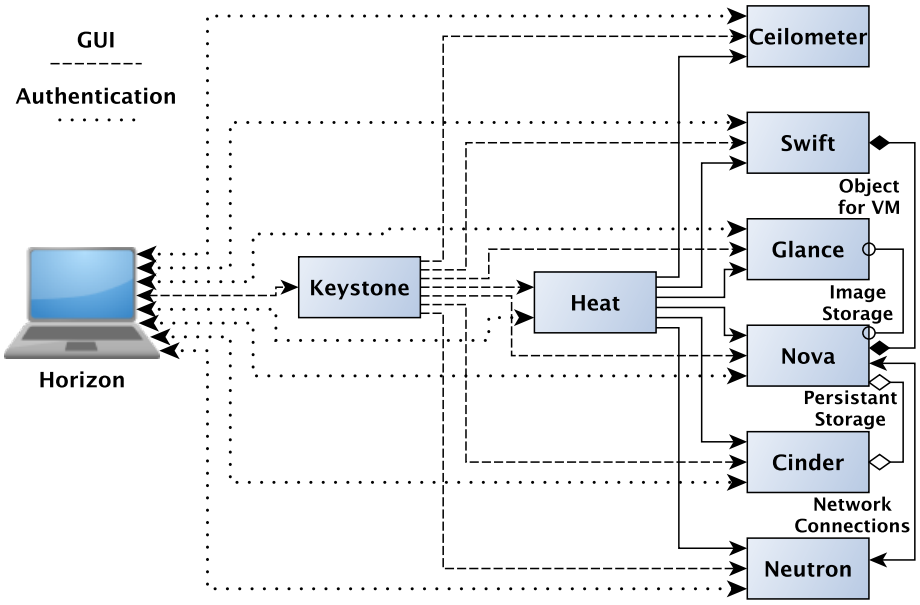

Figure 2.5: OpenStack Architecture

within their cloud provisioned resource. This module is also capable of taking snapshots to enable backing up and restoration of data.

A graphical interface for users and administrators is provided over HTTP in the form of the Horizon dashboard. Horizon allows administrators to manage users, groups and quotas while allowing users to manage their account and directly create and interact with instances.

Identity services are provided through Keystone, which holds a central database of users with mappings for each resource that a user has permission to use. Keystone supports authentication using passwords, token based authentication and Amazon Web Service type logins.

The Glance service manages all aspects of the VM images, including advertising to other services what images are available, the management of snapshots and a users personal images.(Jackson, 2013)

Networking in the latest versions of OpenStack is handled by Neutron, formerly Quantum. This provides a framework which is capable of deploying almost any

network topology to instances through the use of software defined networks, while also handling IP address assignment for instances.
The Heat service provides a template based management service, which enables the automatic creation of instance stacks. All usage based components report usage back to the accounting system Ceilometer that provides global billing across an OpenStack deployment. (The OpenStack Foundation, 2013)

2.7.2 Eucalyptus

Eucalyptus in an open source Cloud middleware which maintains API compatibility with the commercial Amazon Web Services (AWS), the architecture for which is shown in Figure 2.6. This is a very attractive prospect for organisations which may need to scale beyond their own resources into the AWS cloud with minimal user or administrator effort.(Eucalyptus Systems, Inc, 2014). Nurmi et al. (2009) states that Eucalyptus was developed to provide a simple modular cloud platform that was designed with popular academic infrastructure in mind. Within this project, consideration was also made for maintaining a user interface which would be intuitive for existing HPC and Grid users, while also maintaing interoperability with those kinds of systems and associated software.

A complete Eucalyptus installation comprises of five individual components, which are briefly outlined below. The Cloud Controller service is central manager for Eucalyptus, gathering information from and delegating work to other services. This also provides the administration and user web based interfaces for interacting with the system. Networking and VM provisioning is handled by the Cluster Controller. This is achieved via a Node Controller service, which runs on the VM capable hosts, to report system capabilities to the Cluster Controller thereby allowing resource allocation decisions to be made and VM requests to be returned. All images and snapshots need to be stored in a location that is available to all

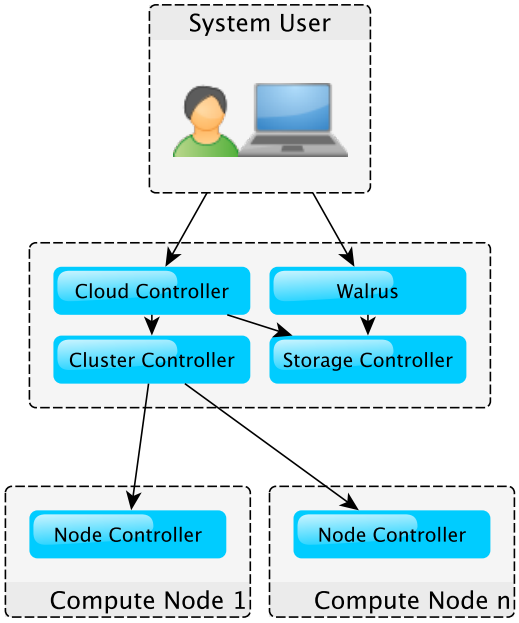

Figure 2.6: Eucalyptus Architecture

components of the system. This is made possible by the Storage Controller which is accessible through Walrus. Walrus provides users with a method for storing persistent data such as image snapshots. This service is also supports Amazons S3 and is compatible with Amazon Machine Image management interface.(Wadia, 2012)

2.7.3 Ubuntu Enterprise Cloud

Ubuntu Enterprise Cloud (UEC) was an Ubuntu branded version of Eucalyptus. This however was stopped in 2011 when Canonical decided to switch to OpenStack for Ubuntu cloud integration (Steven J. Vaughan-Nichols, 2012).

2.7.4 Proxmox

Proxmox is not truly a Cloud middleware in the current sense as it can not scale seamlessly across multiple compute nodes. What it does provide is a management interface to KVM capable resources (Proxmox Server Solutions GmbH, 2013). This

allows VMs to be provisioned in the more traditional way, using standard type installations and whole instance clones. However Proxmox is incapable of dealing with whole system management such as network management or load balancing. It is effectively just a Graphical User Interface (GUI) for standard Linux KVM (Proxmox Server Solutions GmbH, n.d.).

2.8 Grid/Cluster Simulation

A number of existing grid simulation tools, SimGrid (Casanova, Legrand, & Quinson, 2008), GridSim (Buyya & Murshed, 2002a) and Maui's simulation mode exist. These allow potential changes, or entirely new systems, to be evaluated prior to actual implementation.

SimGrid is a tool which provides a comprehensive simulation framework for parallel systems. The main components of this simulator are focused around the testing of new simulation algorithms and evaluation for potential performance of newly developed parallel applications (Quinson, 2009),(Casanova, 2001).

GridSim is a Java based toolkit which allows for discrete event simulation of job scheduling within grid resource brokers (Buyya & Murshed, 2002b). This toolkit, with some developed extensions, is capable of testing Quality of Service (QoS) at a grid level. Work has been completed to show that modelling of a system where job re-negotiation may be required when a resource is unable to meet contractual obligations. Such conditions cause violations of Service Level Agreement (SLA)s potentially causing avoidable financial burden. The GridSim toolkit allows these situations to be avoided by generating results which can provide better constraints for SLAs at the time of creation, making breaches of such agreements more unlikely.

The Maui simulator is capable of evaluating the behaviour of differing Maui policies or hardware configurations. This is achieved through the use of workload and

resource trace files. However the behaviour of this simulator is not very transparent, making it difficult to evaluate the results it generates. Upon completing a simulation the Maui simulator processes data to give metrics presented in % of system efficiency. (Adaptive Computing, 2014)

2.9 Hadoop

Hadoop is defined by Lam (2010) as a framework, developed as open source, for the storage and processing of distributed internet-scale data, more commonly referred to as 'Big Data'. Hadoop is essentially made up of two components. The storage mechanism which handles data within a Hadoop cluster, called the Hadoop Distributed File System (HDFS), and the data processing component of the system which is Map/Reduce White (2010). Hadoop is essentially a cluster middleware, utilising Java Virtual Machine (JVM), specifically developed to process extremely large datasets. A Hadoop comprises four components:

- JobTracker - A Master control node which deals with job scheduling and submission
- NameNode - A HDFS Controller node responsible for data storage and management
- TaskTracker - Compute elements which deal with the actual processing of data within the HDFS.
- DataNode - Compute elements which have storage that forms part of the HDFS.

TaskTrackers and DataNodes are commonly combined into a single piece of hardware, allowing data to be processed by the same node upon which it is stored. (Chandar, 2010)

The HDFS is based upon the proprietary Google File System (GFS). This file system functions by splitting all data into blocks of a pre-determined size, which are than distributed throughout the system. During distribution these blocks are also replicated in triplicate, and each replica is stored on a different node. Thus giving the HDFS very good inherent resilience to hardware failure (Rajaraman & Ullman, 2012). Java is used to provide the Map/Reduce system within Hadoop, This is also based around proprietary work by Google outlined by Lam (2010). As suggested by the name Map/Reduce actually consists of two distinct processes. The Map process takes input data in the form of key/value pairs and performs a user defined operation upon that data. The output from this stage is considered intermediate data and is collected by the Reduce function which consolidates all Map output to generate final results Dean and Ghemawat (2004). A basic overview of this process is shown in Figure 2.7

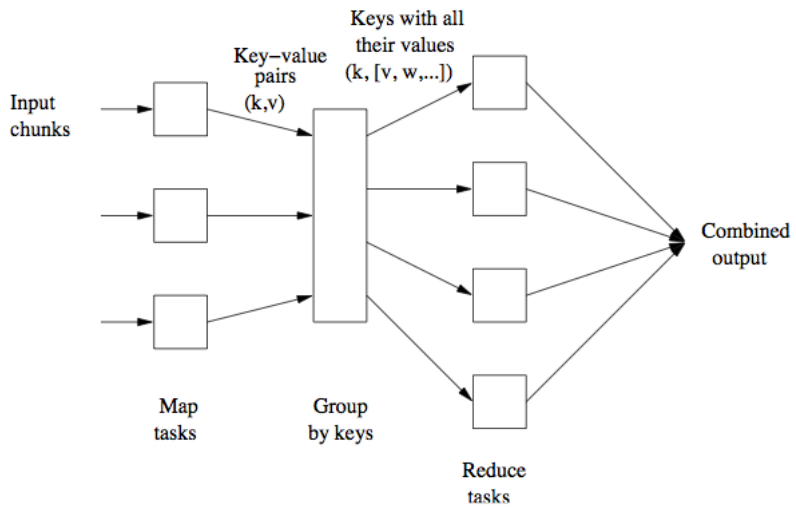

Figure 2.7: Basic Map/Reduce Workflow. (Rajaraman & Ullman, 2012)

2.10 Summary

In this section we have considered multiple grids and grid middleware in use across the globe. It has been shown, particularly with the example of the UK NES, that funding for national and international grid systems can be difficult to maintain. This is an issue which seriously needs addressing, either by somehow creating a much more stable funding model (European Commission, 2013) or, by developing a global flexible grid framework to ensure middleware interoperability without the need for centrally controlled projects (Foster, Kesselman, & Tuecke, 2001).

These large scale projects generally aim to promote the use and development of middleware which is highly interoperable (Kranzlmller, Lucas, & ster, 2010) and sustainable. However, when centralised projects come to an end there will be concerns that the software developers are likely to be unconcerned about the interoperability of software they do not own.

The institutions, who had a previous commitment to a nationally or internationally supported middleware, may be left in a position where it is necessary to maintain excessively resource intensive services. This point will be considered further in chapter 3 using the example of the systems which were maintained but essentially unnecessary after the end of the UK NES funding cycle.

Work carried out on grid systems over the past 20 years has paved the way for cloud computing. This allows resources to be utilised in a much more efficient way than ever before. However, cloud systems have not maintained close links with current grid infrastructure and there are no standard middleware tools to make these systems function together seamlessly. Such a middleware, allowing users to access classic grid systems and cloud infrastructure as a single resource, would allow for global compute power to be used much more efficiently.

Considering Table 2.1 it can be seen that in many campus grids examined,

Campus Grid	**Middleware**	**Access**	**Current Support**	**Easily Scale**	**Admin Effort**	**Source**
Reading	HTCondor	Key	Y	Y	Med	Open
Cambridge	HTCondor	Key	Y	Y	Med	Open
Oxford	VDT	Cert	N	N	High	Open
Manchester	GT	Cert	Y	N	High	Open

Table 2.1: Middleware Comparison

HTCondor is the middleware of choice for many, as it offers a relatively simple scaleable resource. HTCondor middleware is open source and is actively maintained, which makes this choice a very cost effective solution. However it should also be said that HTCondor is not an appropriate solution for all institutions.

Chapter 3

UoH Campus Grid

In this chapter we will consider the current deployment of the University of Huddersfield (UoH) QueensGate Grid (QGG) looking at the hardware which is available along with deployed middlewares and campus cluster systems design. The rationale behind the decisions made on the QGG system design considered here were discussed in chapter 2. There will be an overview of the various possible user workflows within the system along with the level of administrator interaction which is required for each of these.

3.1 QGG Systems

Table 3.1 below gives an overview of the systems which are currently available at the UoH along with a basic information for each system.

As can be seen in Table 3.1 almost all systems in the QGG use Community ENTerprise Operating System (CentOS) which is essentially a rebranded clone of Red Hat® Enterprise Linux® (RHEL) along with a Windows® based Graphical Processing Unit (GPU) cluster. The High Throughput Condor (HTCondor) system is not exclusively a Linux based system, as while the head-node runs CentOS 6

Compute Element	Operating System	Cluster Middleware	Grid Enabled	Num Nodes	Cores Per Node
Tauceti	CentOS 6	Torque/Warewulf	YES	4	4
Eridani	CentOS 5	Torque/VDT	YES	37	4
Sol	CentOS 6	Torque/Warewulf	YES	64	4
Condor	CentOS 6	HTCondor	YES	1300	4-8
Cloud	CentOS 6	OpenStack	NO	2	24
Vega	Windows®	Windows® HPC	NO	3	896 (CUDA)
Bellatrix	CentOS 6	VDT	YES	N/A	N/A

Table 3.1: QGG Overview

the compute nodes are a mixture of Windows® and linux systems, which we will consider in further detail below. All 'grid enabled' systems have at least a minimal Globus installation with some mechanism for publishing resource information. Figure 3.1 describes how the various Linux based systems are interconnected, forming the basis of the QGG.

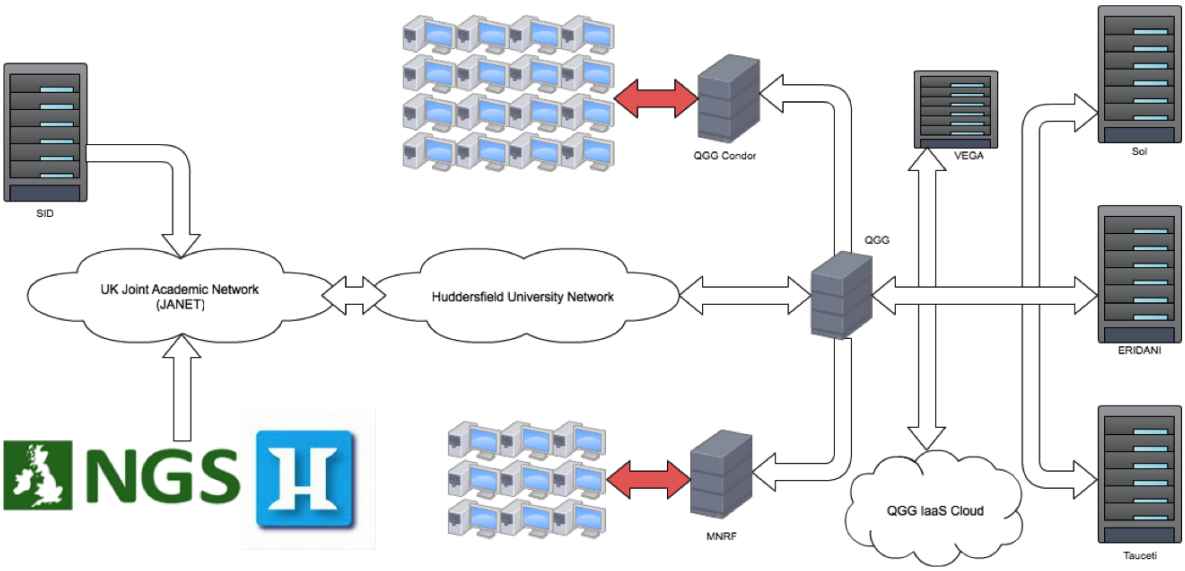

Figure 3.1: QGG Current Configuration Overview. (Holmes & Kureshi, 2013)

3.1.1 Tauceti

Tauceti is a development system and is only included here for completeness. Tauceti runs on the CentOS 6 operating system and uses a Torque based cluster middleware system deployed through the Warewulf Cluster Manager. The Tauceti system is not available to standard users of the QGG and is only used by the administrators to test the deployment of new software.

3.1.2 Eridani

The Eridani cluster is the oldest production server available at UoH and is a true Beowulf cluster built using off the shelf consumer commodity components. The middleware cluster was deployed using Open Source Cluster Application Resources (OSCAR) on CentOS 5, while the grid deployment was handled through the Virtual Data Toolkit (VDT). This is essentially a Torque based cluster which has a Globus gatekeeper for grid based submission, Metacomputing Directory Service (MDS) to publish resource information and Local Credential Mapping System (LCMAPS) plugins to allow for Virtual Organisation (VO) support.

3.1.3 Sol

Sol is based around Sun Microsystems rack mount servers. It is the most recently re-deployed system available within the QGG and has a software deployment similar to Tauceti. Torque is the primary middleware which is deployed through Warewulf onto CentOS 6, and each node is deployed in a stateless configuration at each boot, where the operating system is held entirely in memory. This method, while consuming more memory than a stateful node, allows the administrators to easily re-deploy the system and ensure the base Operating System (OS) of the nodes can never be seriously corrupted. The grid deployment for Sol includes

the addition of Grid Security Infrastructure (GSI)Secure SHell (SSH) and GSI File Transfer Protocol (FTP) to allow communication with more capable servers, as well as an instance of MDS to advertise resource availability.

3.1.4 HTCondor

By far the largest system in the QGG, by node and Central Processing Unit (CPU) count, is the HTCondor system. This system is a truly heterogeneous pool utilising the resources from various different hardware configurations and differing software stacks. Within the HTCondor pool there are around 1160 Windows® based systems, each having between 4 and 8 slots depending on how many CPU cores the individual system has. These systems are located in the University's computer laboratories, and on resources provided by Computing & Library Services.
In addition to the above mentioned resources there are around 140 systems which have Pool Of Virtual Boxes (POVB) installed, allowing linux based jobs to be run on the system. These systems only advertise a single slot within HTCondor regardless of how many CPU cores are available, although all systems are known to have a minimum of 4 cores. Therefore it is possible to run some limited parallel code on these nodes. At the UoH the majority of work carried out requires tightly coupled Message Passing Interface (MPI) networks, which HTCondor can handle but is not the developers primary focus. In situations where MPI is of high priority it is more appropriate to use systems such as Portable Batch System (PBS) and Sun Grid Engine (SGE), as the primary focus for the campus infrastructure.

3.1.5 Cloud

The QGG also provides Infrastructure as a Service (IaaS). This was deployed in the form of an Red Hat Distribution of OpenStack (RDO) based cloud infrastructure

allowing users to spawn virtual machines through a web interface or an Application Programming Interface (API). This system is largely under utilised but has been used for OS based training, to give students experience with system administration. The system has also been utilised for a novel method of automated formative assessment of student work through the use of JClouds and Chef, as described by Bonner et al., 2013.

3.1.6 Vega

This system is a Windows® High Performance Computing (HPC) based GPU cluster, which is primarily used for visualisation. Vega is not currently tightly integrated into the QGG as access is granted through the UoH Active Directory (AD) rather than through the QGG Lightweight Directory Access Protocol (LDAP).

3.1.7 Bellatrix

This is the primary connection point for the QGG. It is a single system which exists within two networks. Publicly it is seen as qgg.hud.ac.uk and internal to the UoH it is seen as bellatrix.hud.ac.uk. Bellatrix is responsible for managing all user connections and for routing any cluster traffic which needs to reach external servers.

3.1.8 GlusterFS

Common home directories for all users on the QGG is achieved through the use of the GlusterFS network file system. GlusterFS provides replicated storage over a local network, removing single point of failure risk (Beloglazov, Piraghaj, Alrokayan, & Buyya, 2012). Through the use of GlusterFS the QGG has two mirrored storage servers located in separate data centres on campus. All systems have access to

both of these servers. The closest of which is configured for primary access. With the further one being used as redundancy. This allows the QGG to provide resilient and reliable storage to users.

3.2 QGG Workflow

Through various modifications completed on the QGG systems there has arisen a situation where a user has various options for their choice of workflow. In this section we will look at each of the possible workflows that are available to users of the system. It is important to note that all Linux based systems within the QGG, with the exception of the cloud resource, have common home directories for all users. This common space is provided through two GlusterFS mirrored systems.

3.2.1 Standard Workflow

The current 'standard' workflow, shown in Figure 3.2, within the QGG requires users to connect internally to Bellatrix.hud.ac.uk, or to qgg.hud.ac.uk if connecting from a remote location. Internal connections are facilitated through the use of SSH keys and all user information is held on an LDAP server.
External connections are made through GSISSH using e-Science certificates. Once the users have been authorised, their Distinguished Name (DN) is mapped to a local account through a grid-mapfile, the information for which is again stored within an LDAP server. The external QGG connection also maintains VO support through an LCMAPS module, however this is only currently used for the DTEAM and OPS VOs to allow for European Grid Infrastructure (EGI) and Grid Operations Centre DataBase (GOCDB) system monitoring.

As shown in Figure 3.2, after the users have been authenticated and authorised on the QGG, they can decide which service they need. This assumes that the

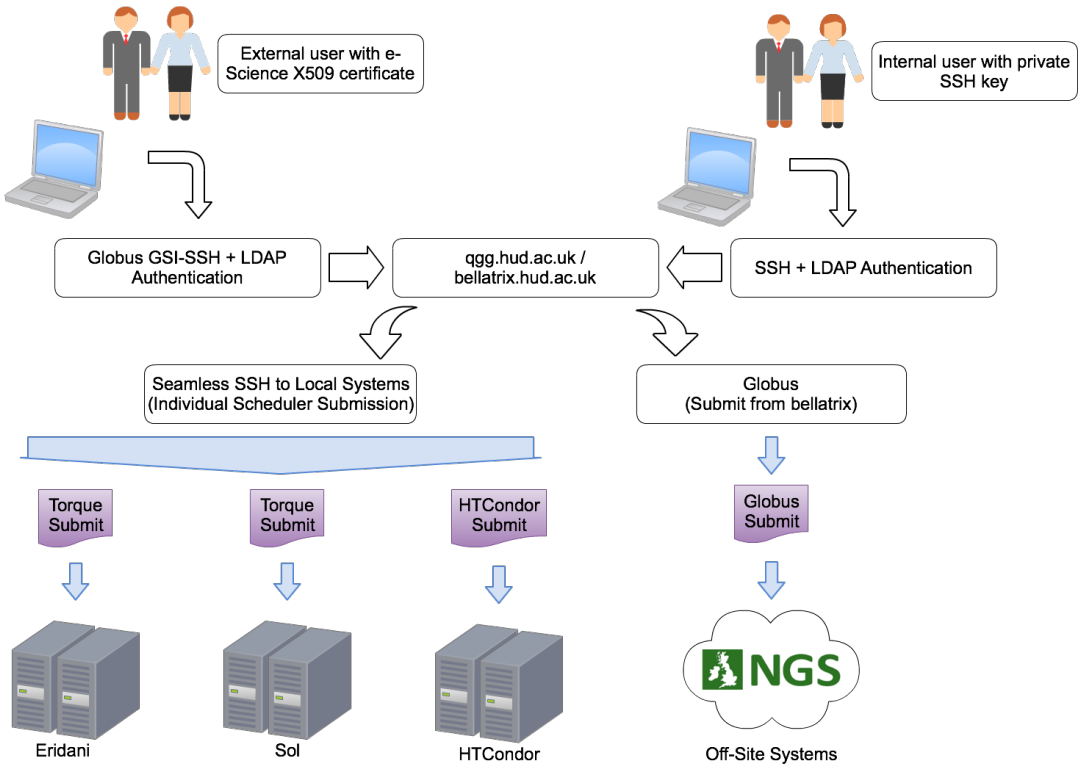

Figure 3.2: Standard QGG Workflow

users need to be aware in advance of the systems available within the QGG so that they can connect to them through SSH. Alternatively, jobs can be directly submitted through use of the Globus Toolkit (GT) to systems supporting that method. Bellatrix also has the OpenStack Python API available, allowing the creation of instances within the cloud.

The decision of which resource to use is entirely left to the user. Once connected to the head node of their chosen system the user may then submit their job script, formatted in the correct syntax, directly to the batch system.

3.2.2 EMI WMS Workflow

In 2012 a true grid middleware was deployed on the QGG, in the form of gLite. This was subsequently updated to European Middleware Initiative (EMI) 1 and then EMI

2 as gLite became part of that project. The purpose of this deployment was to integrate, as far as possible, all systems within the QGG and those resources available through the National e-Infrastructure Service [Formally National Grid Service (NGS)] (NES). The workflow for the user when utilising this middleware is much more streamlined than the standard workflow. Once users have connected to Bellatrix they can then make a seamless SSH connection to the User Interface (UI) node. Once connected to the UI a user may then format their job requirements into a standardised Job Description Language (JDL), which can be submitted to the system. The UI then queries the Workload Management System (WMS), which in turn queries the top Berkeley Database Information Index (BDII), to determine which resources are available to a user based on their VO membership. Information in the top BDII is available because the information has been collected from each individual resource Grid Index Information Service (GIIS), formally MDS, by the site BDII which is in turn queried by the top BDII. The top BDII also queries external site level BDIIs to gather information on which services are available externally for example through the NES.

While the EMI middleware was very flexible, it also had some significant limitations. Integration with HTCondor is not fully supported, and Cloud infrastructure is not supported at all. Administrators at the UoH attempted to integrate HTCondor with the EMI WMS through the use of a Globus Gatekeeper. These attempts failed due to the configuration of the university network. When using the WMS all worker nodes must be able to 'globus-url-copy' the results of any jobs back up to the WMS server itself, from where a user of the UI could retrieve the data. However due to UoH firewall policies this was not possible. Therefore while HTCondor nodes could in fact process jobs, users were unable to retrieve their results. Consequently, the only internal resources available through this middleware were the Torque based

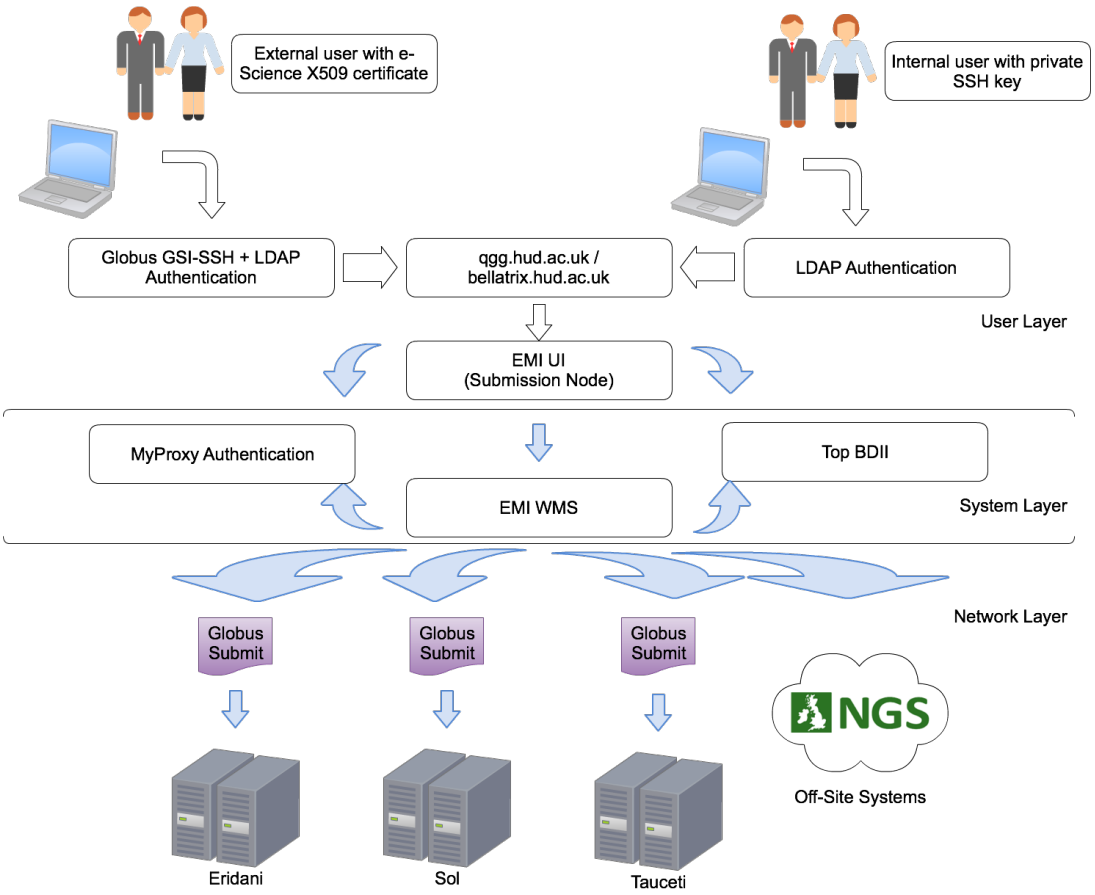

Figure 3.3: EMI Workflow Within the QGG

clusters, although it did simplify submission to a large number of external services. Which means that this particular system was only functional for around 19.6% of the available compute cores on campus.

The EMI middleware controls access to resources based on VO membership, which in turn requires a user to posses a trusted X-509 certificate. This causes a problem within an academic setting, because the UK e-Science Certificate Authority (CA) does not allow certificates to be issued to students. Therefore to provide a fully functioning system available to all users, staff and students alike, it would be necessary to deploy a UoH CA and Virtual Organisation Management System (VOMS) server.

These systems were considered for implementation at the UoH but a number of

factors prevented this. Uptake of the WMS workflow was minimal among the users who did have personal X-509 certificates. This was assumed to be due to the fact that it would require learning a new JDL and changing from a workflow they were already comfortable with.

Another more pressing consideration was the number of servers required to run these services. To provide a fully functioning EMI WMS system, that can be used internally and externally, would require the deployment of six additional servers to those present in the standard workflow as shown within the EMI infrastructure in Figure 3.3. These additional machines must be always powered on, consuming a significant amount of power for little return, detracting from the universities green agenda, and the cost of servers was also a factor to consider. The primary driving factor for maintaing this resources required for the EMI was integration with the NES, but when funding for the project ended in mid 2013 there was no external infrastructure to which the WMS could connect, so becoming a purely internal system. The system is still workable through the gridpp VOMS provision, but remains largely unused due to the factors discussed above.

3.2.3 HTCondor Workflow

Users can also elect to connect to the QGG HTCondor head node and submit all jobs from there. HTCondor defines jobs in subsets, or as the middleware terms it a job 'universe'. The grid universe is available, allowing jobs to be submitted to any resource which has a Globus gatekeeper. Therefore all job submission can be achieved using a single JDL. The main drawbacks for this process are that HTCondor has no real mechanism for automated discovery of grid resources, and as stated previously, users are reluctant to move away from the familiar Torque based submission. The real issue with not a having resource discovery mechanism is that a user must specify which endpoint they would like to use in the

JDL. The users would need prior knowledge, and in this case could just as easily make a direct SSH connection to the known system instead of via the HTCondor head node. The real power of HTCondor lies within the *vanilla* and *standard* universe models. Within the vanilla universe HTCondor will choose suitable resources from the pool of discovered resources based on the requirements of the job. The standard universe is similar to vanilla, but has the added functionality of job checkpointing. This allows binaries that have been compiled with *condor_compile* to periodically checkpoint their progress. Progress information created in this way is sent to a checkpoint server, which can subsequently be used to restart the job at that point if it fails for any reason, enabling HTCondor to deliver an extremely resilient resource.

3.2.4 Globus Workflow

The GT is also installed on all Linux based systems within the QGG. This is configured to trust the UK e-Science CA, but does not currently have any VO support, and all trusted users are defined within the grid-mapfile. Bellatrix does not have a globus-gatekeeper installed. Consequently, any external user cannot directly submit jobs to the QGG. Therefore to submit a job using the GT a user must already be connected to the system. The additional complexity of generating proxies etc, means this system is only really used within the QGG for training purposes.

3.3 Summary

In this chapter, the QGG campus grid was described in terms of systems, hardware and software. Set out how systems architecture decisions are made based on the properties of available Grid middleware. Within this context a number of workflow models were explored. While the standard workflow allows almost all systems to

be accessed from the Bellatrix node it does depend on the users knowledge of the QGG. Without information on what systems are available and the related batch system, published on the QGG webpage, it would be very difficult for a user to navigate around the QGG.

The EMI middleware is a powerful tool capable of resource discovery and load balancing. However the requirement of X-509 certificates, along with VO memberships, presents a great deal of complexity to users and administrators. In addition, the students are unable to gain certificates from the nationally recognised CA, and would be prevented from using the system.

.

The standard model remains the workflow of choice for users, most of whom elect to exclusively use the Torque based systems, with no apparent regard for which system is actually the most appropriate for the job. This issue can be addressed by moving the decision on where to run jobs away from the users and having the decision made at system level.

Such a change requires that different compute elements interact, to become more autonomous, and reduce a users input. However, when opening lines of communication between systems which did not previously have any system level interaction, consideration must be given to the security impact of these changes. Even within a 'trusted' environment such as a closed campus network, there is always the possibility that data could be compromised. Therefore current security standards should be investigated, considering if or how such security would be used when opening additional lines on communication between systems.

Chapter 4

Grid Security

Within any distributed computing environment security is always a key concern. Wherever systems can communicate potentially sensitive information, steps must be taken to ensure those messages stay private. According to Lee, 2013 there are three cornerstone properties of computing security: Confidentiality, Integrity and Availability. This means that any data within a computer system must be stored or transmitted in such a way that it cannot be accessed by anyone, or anything, that is not authorised. The data must not be modifiable by any subversive means and data must be maintained, while remaining available to any sufficiently authenticated and authorised agents. The most common methods for ensuring the privacy of computer communications, are using some form of cryptography to encrypt all traffic that passes between machines. Two of the most common forms of encryption are considered in this chapter. (Lee, 2013)

4.1 Access to Campus Grids

Campus or national grids generally use one of two forms of encryption in order to secure traffic. A long standing method is to use Digital Signature Algorithm (DSA)

or Rivest-Shamir-Adleman (RSA) encryption to provide private keys to be used for symmetric key cryptography, often referred to as Pre-Shared Key (PSK). As grids have developed another standard has increasingly been adopted, using the same encryption methods, in the form of asymmetric key cryptography underpinned by Public Key Infrastructure (PKI).

4.1.1 PSK

PSK security is based around symmetric key cryptography. Using this model the same key is used to both encrypt and decrypt data. This means that each party that needs to have access to any given data, must have the same key. This key must somehow be shared before any encryption can be used.(Wilkinson, 2010)

4.1.2 PKI

PKI is based around asymmetric key cryptography. One key is used to encrypt data, which only the owner should have access to, and is referred to as the private key. Another key, the public key, is made available to all and is required to decrypt anything encrypted by the private key. Using this method the reverse is also true, where data encrypted with the public key can only be decrypted by the private key. There is a problem however, because the public key is available to all, the origin of anything encrypted with it can not be guaranteed. Similarly just because someone has a private key their actual identity cannot be assured. The first of these problems is solved by both parties having public and private keys, each using their own private key to encrypt data. The issue of proving an individual's identity is solved by introducing a third party, in the form of a Certificate Authority (CA). The CA is responsible for ensuring a purported identity is actually valid. With this assertion made, the CA generates a certificate which assures who has

the ownership of a public key. This allows two entities to communicate, trusting the other party they are communicating with, so long as the integrity of the CA is maintained. (Wilkinson, 2010)

4.2 Security in the QueensGate Grid (QGG)

Connection to the QGG supports both PSK, through Secure SHell (SSH) and PKI, through Grid Security Infrastructure (GSI)SSH. SSH is, almost, exclusively used internally, and through the use of PSK allows seamless connections between the various resources available on campus. Until recently (2013) GSISSH was the only supported method for connecting to the QGG from any system that was external to the University local network. The certificates which were accepted through this method were signed by the UK e-Science CA, with users being mapped to static or pool accounts based on their existing relationship with the University of Huddersfield (UoH). This allowed absolute confidence in the identity of connecting users, and also allowed users the same confidence in connections made to required services. In late 2013 the policy to only allow GSISSH for external connections was relaxed and an SSH service was made publicly visible. The main driving factor for this shift was due to the fact that students of the UoH, or indeed any university, are not eligible to obtain a UK e-Science CA signed certificate. Additionally, even for users who are eligible to receive a UK e-Science CA certificate, GSISSH involves a greater level of complexity then SSH does. This presented a situation where many of the QGG local users could only access resources when on campus. It was decided that an SSH daemon, accepting connections based on PSK, would maintain security while allowing all validated users to access the system when off campus. To mitigate the inherent problem of PSK, the sharing of the keys, when a user is created the generated keys are only available to the user within the trusted local

network. Regardless of the method used to connect to the QGG, all user information is held within an Lightweight Directory Access Protocol (LDAP) server, moving user account control away from accessible systems. This provides additional security, by ensuring no user ever directly connects to the machine responsible for user management, and it allows User IDentity (UID)s and Group IDentity (GID)s to be common across all QGG systems.

4.3 Summary

Security within any computational environment must be given a great deal of consideration. The main emphasis of this for system administrators is to ensure that only authorised users are allowed to connect to a system, and that those users connot manipulate other users data or connections. Where connections are to be made externally from a potentially unknown source, such as a foreign collaborator, PKI along with a mutually recognised CA provides an invaluable security tool. However, this does create a higher level of complexity for administrators and users alike, within institutions where a great deal of control can be leveraged around the creation of user accounts and keys. So long as the level of encryption on the keys is sufficient, PSK over SSH can be considered a suitable security measure. However, care must be taken to internally assure the identity of any potential user, and even then those credentials could never be trusted by any external network.

Chapter 5

PBS Based Trusted Grid Development

5.1 Introduction

As mentioned previously, where multiple resources are available, load balancing becomes very difficult as users have a tendency to consistently use the resource they are most familiar with. For many reasons, such as time, cost, etc, it is always preferential to balance load across all suitable resources. One approach to solving this would be to force all users into a particular batch system, such as High Throughput Condor (HTCondor), which is capable of submitting to all resources. This option was dismissed as some resources would still have to be explicitly specified by the user, and it was anticipated that users would be unhappy with this approach. Therefore a submission system based on the 'favourite' batch system needed to be devised. Such a system needs to provide a submission mechanism which behaves, from a user perspective exactly the same way as Torque, while allowing the administrators to control load balance by sending jobs to an appropriate resource. Another important aspect, from an administration perspective, was

centralisation of all accounting records. Such a provision is essential to ensure all resource usage is attributed to the appropriate user. Occasionally when users require resources which exceed those available within a classic cluster environment, it was considered that use of a cloud based torque node would be the preferential method of handling such requests.

5.2 Development

In order to achieve all of the previously stated requirements for this project, it was deemed necessary to separate the problems into two distinct subsets. This was required because allowing Torque to submit to HTCondor involves making the decisions after the submission of a job. Inversely, in order to provision resources which are not available, the job submission must be intercepted so resources can be provisioned before Torque checks to see if such resources exist. The first of these problems was addressed in implementing the *PBStoCondor* project with the latter being handled by the *pbsSurge* code. The *PBStoCondor* software and *pbsSurge* scripts are novel solutions to the problems faced, and were devised during the course of this project. As the problem presented was considered to be essentially a scriptable problem, but beyond what was feasible for a standard shell interpreter, the decision was made to implement it in Python. Python offers high execution speeds when compared to PHP or Ruby, has good shell integration (McKinney, 2012), and of particular interest for this project, Python's direct Application Programming Interface (API) compatibility with OpenStack.

5.2.1 PBStoCondor

The first problem to overcome for this project was enabling Torque to recognise the Pool Of Virtual Boxes (POVB) based resources within the HTCondor pool, as potential compute nodes. It was determined very early in the project that it would not be possible to simply run a pbs_mom on each of these nodes, as such an arrangement would be placing jobs on HTCondor resources without any of the HTCondor services being aware of them. Therefore all jobs needed to be run through HTCondor but monitored by Torque. From Torque version 3.0 it has been possible to run multiple moms on a single node. With this capability it was possible to run as many moms as there were required. However, consideration needed to be made, of how much impact running many additional daemons would have on the HTCondor head-node within the QueensGate Grid (QGG) (for example there were 200 POVB nodes). Through observation of running moms it was found that an average mom uses minimal ($<$0.1%) Central Processing Unit (CPU) time and around 30MB of memory. The HTCondor head-node in the QGG had a total of 15.5GB of memory of which 13.8GB was free with all required processes running at any given time. This means that the system was theoretically capable of running 471 moms, 235% over the potentially required 200 moms. The most intuitive method for allowing Torque to choose POVB nodes was simply through queue management. Torque was configured to send all jobs which requested a serial queue, and a short enough wall-time, to the HTCondor based moms, where wall-time is the total amount of real time taken up by a job. The reason for the wall-time stipulation was that common applications were being used across all *NIX systems, meaning jobs were limited to the vanilla universe. Therefore it was undesirable to have very long running jobs submitted to HTCondor resources. To handle submission into the HTCondor batch system a National Grid Service (NGS) Universal Execution Environment (UEE) style of application naming was adopted. This entailed

the creation of a '/ngs' folder on all systems which would hold symlinks specific to each system. On a pure Torque system, and on the POVB nodes, these simply pointed to the relevant application. However, on the HTCondor head-node, the symlinks pointed to a script which would generate a job submission script suitable for HTCondor and subsequently submit it to the system. This script was also required to monitor the progress of the job and modify Torque accounting records to maintain accuracy of the accounting across all systems. The script developed is contained within Appendix B. The basic workflow of this system is quite simple, as described in Figure 5.1.

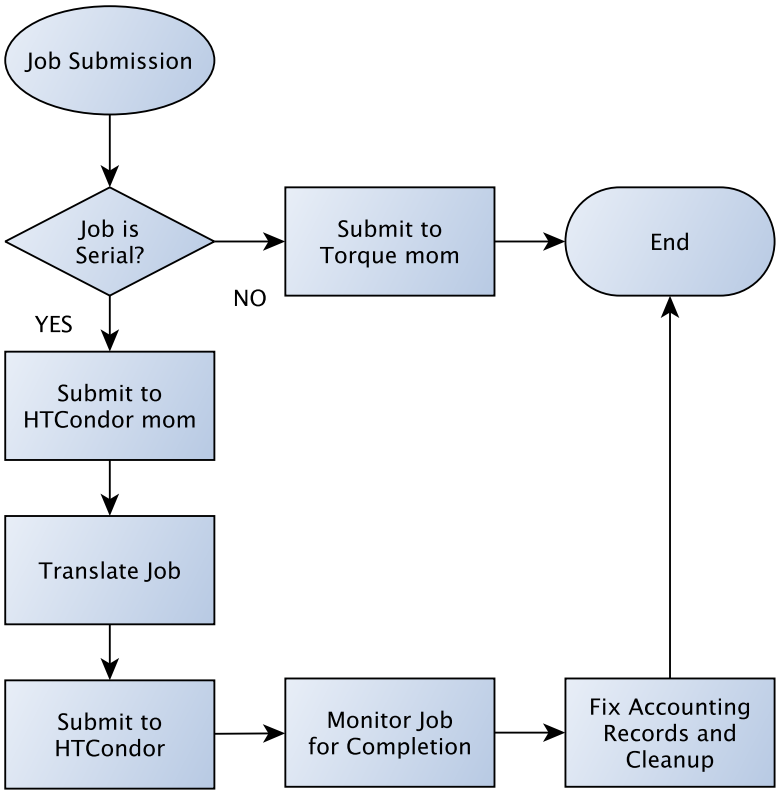

Figure 5.1: Proposed System Overview

Although this system worked, it relied on Torque moms being manually started on the HTCondor head-node. Given the dynamic nature of HTCondor this was not a feasible solution. To address this issue a daemon was developed to monitor the

available resources and start moms accordingly.

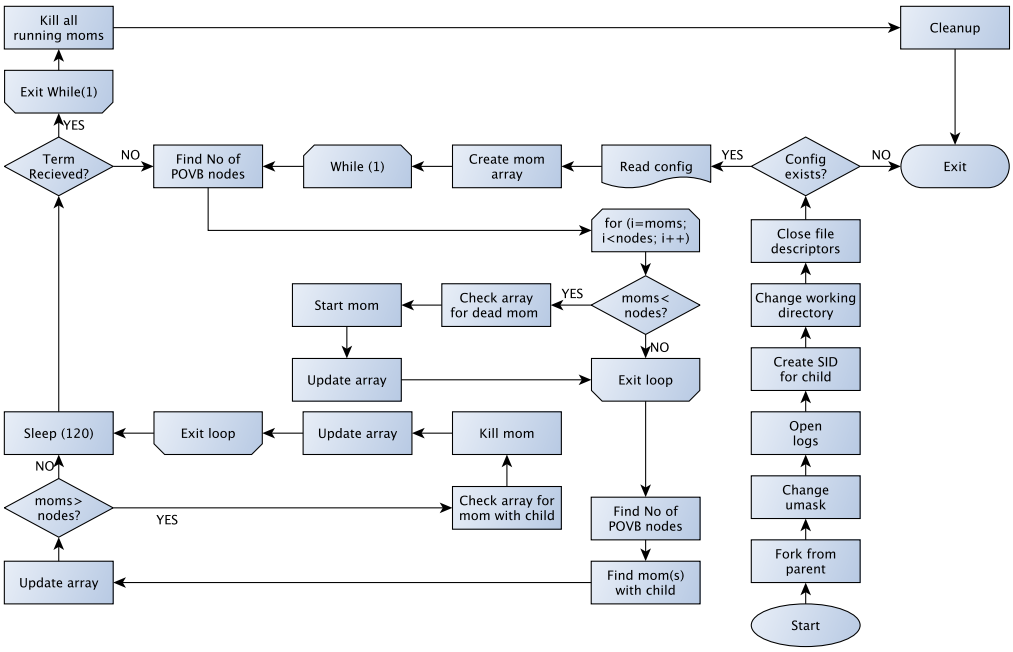

Figure 5.2: Flowchart for daemon

In Figure 5.2 the functionality of the developed daemon is shown, (the code for this daemon can be found in Appendix C.) The daemon is started as a standard system service, which will fail if the configuration file is missing. The initial startup procedure creates a table of possible moms, including name and required ports, generated from the variables within the configuration file. Once the program is running in the background, it cycles through the while loop once every 30 seconds. This was done in order to prevent the daemon from unnecessarily using CPU cycles. HTCondor pool is queried to discover the current number of POVB nodes which are in an 'unclaimed' state. If this number is greater than the number of moms currently running, then an appropriate number of moms are started. If the number of 'unclaimed' nodes is less than the number of moms that are running then a suitable number of moms are killed. When this situation is true, the daemon checks that the mom to be killed has no child processes. This ensures that a mom which is currently processing a job will not be killed by the daemon. If the mom

currently being evaluated has child processes the daemon will then consider the next mom within the array. Every time the daemon starts or stops a process, the internal data array is updated to ensure there is always valid information of which moms are running for the system to query. If at any point the daemon receives a termination signal before starting a new cycle it will end all running moms before exiting. This will ensure there are no orphaned processes on the system. *PBStoCondor* handles all short serial jobs submitted to a Torque batch system which can be pushed to a more appropriate HTCondor resource. However, as mentioned previously, there was also a requirement to allow the system to provision for jobs which required resources greater than those currently available. This is considered in the next section.

5.2.2 pbsSurge

When a job is submitted to a Torque based system which requires more resources than the current configuration can provide, the job is immediately refused by the scheduler. Therefore job requirements needed to be checked prior to Torque submission. The *pbsSurge* script was developed to handle these types of request.
Figure 5.3 shows the decisions and actions taken by the code which can be found in Appendix D. This functionality is actually provided by two separate scripts. The first script, referred to as the *submitter*, usurps the standard *qsub* command and parses the job script for the requested resources. If these resources can be handled by the system in the current configuration, the job is immediately passed on to the standard scheduler. If the requested resources do exceed what the system can provision, then the *submitter* takes over for a time. The process must first determine, through the use of the OpenStack API, if the configured cloud has enough resources to provision for the job. If it does not have sufficient resourcing then it immediately exits reporting an error back to the user. In case that the configured

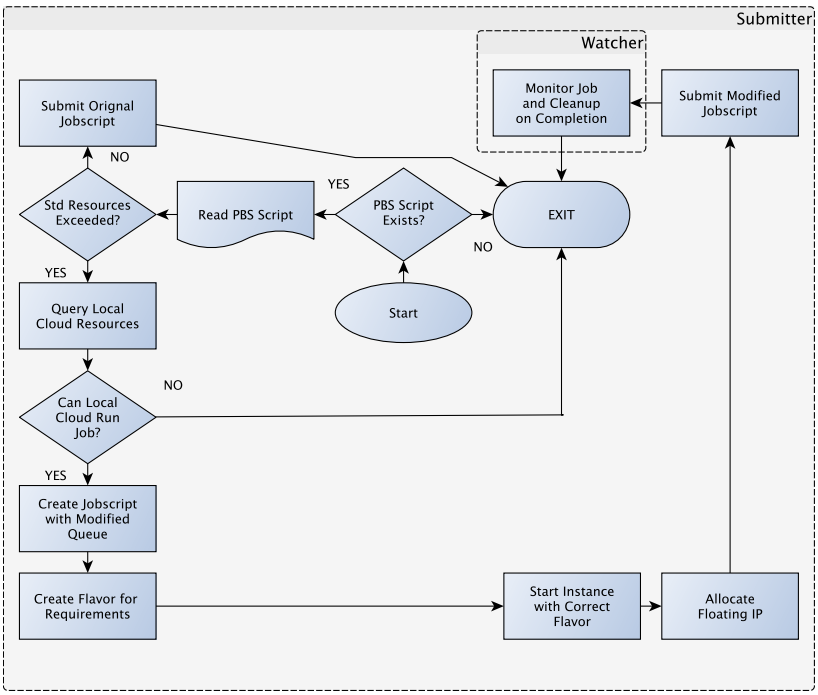

Figure 5.3: Flowchart for pbsSurge

cloud has sufficient resources the first script runs to completion. Initially the Torque job script is modified to change the requested queue to one associated with cloud deployed nodes. The script then generates a custom flavour (OpenStack Foundation, 2013a) within the cloud, based on the job requirements, and creates an instance using the created flavour using an image (OpenStack Foundation, 2013b) which has been previously configured to function as a Torque node. Once the instance has finished spawning, and has been allocated an IP address, the job is passed back to Torque and scheduled as normal. The *submitter* acquires the job information at Torque submission time, passes this information to the *watcher* script and exits. The *watcher* periodically queries Torque to discover when the job has completed. Once this happens the *watcher* destroys the cloud instance and deletes the flavour which was created for the job.

5.3 Deployment

The developed systems were designed to be simple to deploy. *PBStoCondor* has been packaged into Red Hat Package Manager (RPM) packagess to allow for rapid deployment on any system which uses RPM packages. *pbsSurge* only needs a symlink called *qsub* within the path the supersedes the standard Torque *qsub*. Both require modifications to Torque, creating a dedicated queue for each. The only system dependancies for these are Python2.x and the OpenStack client tools.

5.4 Summary

The two developed systems allow Torque to delegate particular jobs to a HTCondor system and to move very large jobs into the cloud. Most importantly, this added functionality allows for better use of more tightly coupled resources while maintaining a consistent, and familiar, submission method to users. While the *pbsSurge* code performed well in testing the true benefit to the system is difficult to quantify. This is due to the fact that users are aware of the system limitations and remain within those constraints or find a larger system. It was deemed that quantifiable results could only be gained by deploying this system and making users aware of it, which was not possible at the time of this work. As the behaviour of the *PBStoCondor* system is very predictable, there was the opportunity to test this system by using historic Torque accounting logs. The aim of which was to highlight improved overall throughput by automatically pushing all serial jobs to HTCondor.

Chapter 6

Simulator

6.1 Introduction

In order to test the effectiveness of *PBStoCondor* two potential methods were considered. The first of these methods was to implement the developed system onto the production systems in the QueensGate Grid (QGG). This approach was deemed unsuitable for a number of reasons. The first factor was repeatability of the test results. As user behaviour fluctuates during any given period of time it could have been very difficult to make a meaningful comparison between the system before and after implementation. Another reason was lack of time. Once the system had been implemented, there would have been a very limited record of system behaviour to analyse. During a limited period of a few months, with the aforementioned fluctuations in job submissions, the real-time test results would not have given a fair evaluation on the developed system. Therefore other possibilities needed to be considered. Hence, it was decided to to simulate the system exhibiting the modified behaviour induced by *PBStoCondor*. The goal was to achieve simulated results that would be directly comparable to historical Torque accounting data collected over twelve months. This approach was not without difficulties. A

number of existing grid simulation tools, SimGrid (Casanova et al., 2008), GridSim (Buyya & Murshed, 2002a), and Maui in simulation mode were considered for this task. These were found to be unsuitable. All of the developed grid simulators are targeted at application performance or scheduler algorithms. For all experiments the scheduler behaviour and performance of applications was required to be constant, as determined by historic accounting data. Also, the experiments would only be required to consider changes in the number and configuration of systems, not any differences achieved through hardware or application modification. The key questions that needed to be answered were: can resources be better utilised and can job completion times be improved? As none of the available simulation software could answer these questions, it was decided to design a simulator capable of providing answers to these questions. The work detailed within this chapter was completed in collaboration with Mr Ibad Kureshi, and the Hadoop code described was developed in collaboration with Mr Stephen Bonner.

6.2 Development

Once it was determined that the effectiveness of *PBStoCondor* was to be tested by simulation, the next step was to clearly define required behaviour of the developed simulator. To maintain compatibility with the rest of the developed software, Python was chosen as a programming language. It was decided that the simulator must:

- Create a simulated system of any required configuration.
- Read in job information from historic Torque accounting logs.
- Schedule the jobs to an appropriate simulated resource.
- Remove 'completed' jobs from the simulated resource.

- Produce modified logs for simulated system

Within the brief framework outlined above, specific behaviour needed to be ascertained. A simulated system would be constructed of *x* number of nodes with *y* number of cores. However, it is not required that all nodes have the same number of cores. Hence, the simulator also needed to be capable of simulating a system of *x* nodes where *y* cores could be different for each node. When reading historical logs, only those records pertaining to the completion of a job should be evaluated by the system, as these records contain complete job information. In order to maintain data integrity, the time at which a job was created on the system, referred to as ctime within Torque, should never be modified, otherwise the historical user behaviour would be modified. The simulator need only produce modified records for the end of a job. This output was to be produced based on when the simulator decided a job could start and the historic duration of the job. An assumption is made here that a job will take the same amount of time to complete in the simulator as it did in reality, recorded in the Torque log. This was considered acceptable as the additional resources made available by the *PBStoCondor* tool should enable faster execution of jobs compared to jobs originally run on smaller resources. Some mechanism needed to be employed to keep track of how long a job would be 'running' without it being tied to any clock or considering real-time. The final logs produced by the simulator were required to include all pertinent information gathered from the historical logs such as User IDentity (UID), job ID, time job was created etc. Simulated logs should also provide modified, if appropriate, modified job completion times. All times within these logs should adhere to Unix epoch standard. UNIX epoch standard determines time in the number of seconds that have elapsed since midnight on the first of January 1970 using Coordinated Universal Time (Love, 2005). The scheduling of the simulator was to consider jobs on a first come first served basis, with aggressive backfilling. Implementation of fair share

policies within Maui is difficult to determine. Therefore the decision was taken to initially leave this functionality out of the simulator and analyse what impact this had on the results.

6.3 Runtime Functionality of the Simulator

The structure and functionality of the developed code (full listing available in Appendix E) is outlined in Figure 6.1. The simulator is invoked via the console and requires a Torque log file to be passed as an argument. In the beginning, the program checks for the presence of a 'resources.txt' file in the working directory which describes the resources to be simulated. Using the information in this file a 2D array is generated where rows denote nodes and columns denote cores. Table 6.1 shows a sample array, consisting of 3 nodes each with 4 cores. Cores within nodes are represented in a binary form. A "1" indicates the core is free and a "0" conversely denotes a core which is busy. All cores are initially set to "1", representing an entirely free system. Table 6.1 defines a system where one node has 2 busy cores and the other systems are entirely free. The next stage is to read in the

0	0	1	1
1	1	1	1
1	1	1	1

Table 6.1: Sample Simulator Array

Torque log file and use the information extracted from it to populate an initial table. This initial table is populated with: jobid, user, queue, ctime, qtime, etime, start, end, nodes, ppn and duration. Duration is a value calculated within the simulator using start and end. All variables used are defined in Table 6.2.

The information held within the initial table is then bubble sorted (Daintith & Wright, 2008) based on jobno. This is required because the simulator reads in job end

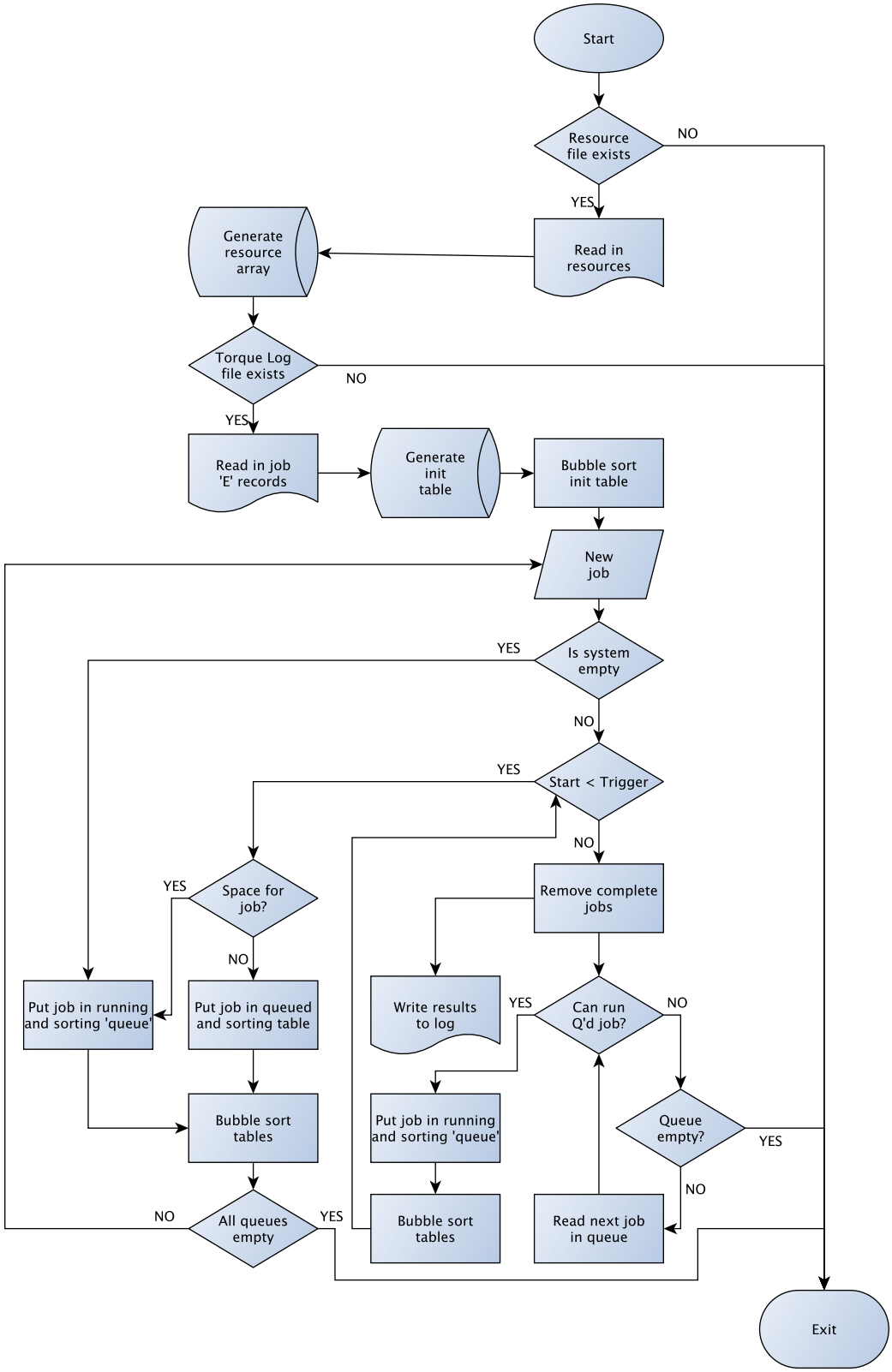

Figure 6.1: Simulator Flowchart

Variable	Description
jobid	Unique job identifier allocated by Torque
user	UID job running under
queue	Queue requested for job
ctime	Time job was created
qtime	Time job was queued
etime	Time job became eligible to run
start	Time job started to run
end	Time job exited (either successfully or unsuccessfully)
nodes	Number of requested nodes
ppn	Number of requested cores per node
duration	end - start
trigger	Earliest end of a running job within simulator

Table 6.2: Torque Log Naming Conventions

records which, due to duration differences, are unlikely to be found in chronological order with respect to creation time. Creation time cannot be used for this sorting procedure as it is possible for more than one job to be created at exactly the same time. As the emphasis is on emulating an existing system it is reasonable to simply use the job numbering that was historically determined. It should be noted that any jobs that require more resources will be excluded from the initial table and an error will be produced to inform the user. The program can now begin 'running' jobs. The system will consider jobs in the initial table one by one. If there are enough free resources those cores will be set to busy and allocated to the incoming job. Then the job will be moved to a running table and marked as such in a sorting table. If the system does not have enough free resources to service a request then the job is placed in a queueing table and appropriately marked in the sorting table. The sorting table holds a value referred to within the program code as trigger time. Trigger time is the mechanism which controls time within the simulated environment. The trigger time is evaluated based on when the job has been allowed to start and the job duration. Effectively this is the earliest time when a job in a running state within the simulator will end. For the first job to enter the system this trigger time

is simply the end time gathered from the original logs. Every time a new entry is added to the sorting table it is bubble sorted based on the trigger time value. The program can then check if the start time of an incoming job is greater than the smallest trigger time within the sorting table. This represents a situation where a job is created after another is finished. Therefore the completed job occupying the system can be removed. A log entry for the removed job can be printed out. The new job can start, with appropriately modified start and end times. When a job enters the system that cannot be immediately run due to lack of resources. That job will be queued and the next job in the initial table will be considered. This continues until all jobs are either running or queued. At this point the program continues to cycle, modifying start times based on trigger times of previous jobs. Determining a new end time for each job that is allocated resources and marked as running. This continues until all tables are empty, at which point the simulator exits. This process provides extensive log files which require some form of analysis to determine exactly how the system has behaved.

6.4 Verification of Simulator

In order to use the simulator in evaluating the *PBStoCondor* system, the operation and accuracy on the simulator first needed to be verified. To achieve this, the simulator was run with combined historic Torque logs produced over one year. During this verification test, the simulator was configured to have the same resources as the Eridani Cluster, the system which originally ran the jobs. The historic logs were limited to one year due to the requirement of the simulator to have sequential job numbers. The simulator running time varied greatly based on the provided system configuration. When running based on the original Eridani configuration the simulation took 1.3 hours to complete. When using a configuration representative of Eridani enabled with *PBStoCondor* the simulator took 3.5 hours to complete. The simulator output and the original logs were then analysed through Hadoop to find the respective arrival and completion rates for each log file. Within this context, arrival and completion rates are defined as a number of jobs arriving or completing within a defined time period. The Hadoop code developed to perform this function is written in Java and can be found in Appendix F. Arrival rates would always remain the same as the simulator cannot modify the time a job had been historically created. The significance of this test was to ensure that results from the system being simulated, matched a configuration produced in the historical logs. Then the job completion rates for the simulator logs and the historical logs should be as close as possible, preferably identical. The produced results were encouraging and will be considered at length in chapter 7.

6.5 Summary

This chapter presents the design and development of a simulator and the considerations that led to its creation. In order to test the effectiveness of *PBStoCondor* it was determined the most appropriate method was to use a simulator. This simulator was required to allow direct comparison between historic and modified High Performance Computing (HPC) system configurations. Output was formulated to closely replicate actual Torque logs to simplify comparison. The primary interest of this endeavour was to consider the impact of removing serial jobs from a parallel system, rather than resource dependant application speed. Jobs were required to be scheduled, to 'run' on a suitable resource, and allowed to 'run' to completion. The developed simulator was able to perform all of the functions that were defined as required prior to its development. The resulting simulator produced a mechanism for testing the impact on job completion times when more resources are added to a known system, assuming that the duration of a job will not change. Also, as applications and hardware specifications beyond number of cores was not considered, but start and finish times were considered to be fluid. The simulator produces very large amounts of data which needs to be analysed for meaningful results. This analysis was achieved through the use of Hadoop, which is specifically designed to handle large amounts of data. Utilising the simulator tool, together with Hadoop, resulted in the creation of structured data and information to allow the behaviour analysis of existing systems. Thereby providing better system understanding to allow for improved future system design.

Chapter 7

Results

7.1 Introduction

Once the simulator and *PBStoCondor* had been developed it was possible to test these two systems in three stages:

1) Initially the simulator code required verification. This was achieved by configuring the simulator to mirror an existing High Performance Computing (HPC) system comparing the output to ascertain that the simulator behaved as like the real HPC system.

2) The simulator could then be configured to behave as a *PBStoCondor* enabled system.

3) Only then the results for the *PBStoCondor* system could be compared against the original system output, hopefully showing some improvement in the rate at which jobs were completed. As mentioned in the previous chapter, the work detailed within sections 2 and 3 of this chapter was completed in collaboration with Mr Ibad Kureshi, and the Hadoop code described was developed in collaboration with Mr Stephen Bonner.

7.2 Testing Procedure

The test strategy was based around one year of Torque accounting logs taken from the Eridani cluster. Torque logs are organised as a single file for each day as they are produced. Using these files a single file of all logs from 2013 was produced, which is referred to as the original Torque log. Initially there was an intention to use three years of logs from the Eridani cluster containing records of jobs created and completed. However, through regular maintenance and re-installation of Eridani there were points where the job ID counter had been reset. This posed a problem for the simulator, as sequential job IDs were required. It was therefore deemed that one year of logged system behaviour would be a sufficient data sample, representative of ongoing system loads. The original Torque logs were then passed to the simulator along with a configuration file determining the system to be simulated. Depending on the system to be assessed simulation time varied greatly, using a 36 quad core node simulation took around 1.5 hours and 36 quad core nodes plus 200 single core slots taking around 3 hours. Once simulated log files had been produced, those logs, along with the original Torque logs, were passed to the developed Hadoop code shown in Appendix F. The purpose of this was to extract useful data, such as arrival and completion rates, from the very large logs. Hadoop was used for this task because it is a framework specifically developed for dealing with large data sets. Through the use of Hadoop Distributed File System (HDFS) and Map/Reduce large amounts of data can be quickly analysed based on a user defined algorithm. Using this method allowed log files for an entire year to be analysed in under one minute. The algorithm used determined how many jobs had been completed within interval x which was n seconds in length. Intervals were all based upon epoch times read from job end times within the logs. For simplicity epoch 0 was set at 2013/01/01 00:00:00. To clarify, using an interval of 24 hours,

or 86400 seconds, produces a log of 365 intervals with an associated Arrival/Completion value. These interval values can be converted back to a standard epoch format to find a real time and date using the following formula:

$(Interval * IntervalLength(s)) + 1356998400$ Where 1356998400 is the true epoch for 2013/01/01 00:00:00. This produced a log of how many jobs had been completed within a given time interval (Completion Rates). Also, in the case of the original Torque logs, the number of jobs created on the system within a given interval was determined (Arrival Rates). These logs required one further processing step. The Hadoop process would not report intervals in which zero jobs arrived or completed. To rectify this a simple script was created to fill in the missing zeroes within the Hadoop output (zero padding), ensuring that results using the same time interval would have matching dataset lengths. This allowed for direct comparison of simulated and real system performance.

7.3 Simulator Validation Results

Using the procedure outlined in the previous section, results were initially obtained to show the validity of the simulator using a known system layout. During 2013 Eridani had 36 nodes, each with 4 cores, and the simulator was configured accordingly. Using the Hadoop cluster, completion rates were calculated for 24 hour, 1 hour and 15 minute intervals. Considering job completion data for the entire year, Figure 7.1 shows the results for the original Torque logs and those for the simulated completion; they are almost identical. Looking more closely at the data for a 1 month period, April 2013, shown in Figure 7.2, it can be seen that in periods of high load there is some discrepancy in the results. This is due to Maui fair share policy having an impact on the real system, since the simulator does not have a fair share feature. This limitation translates to an average difference of 8 jobs per

day between the simulated and original data using 24 hour intervals. This is only 1.2 jobs when considering 1 hour intervals. More detailed analysis of the data considered a 2 week period between March and April 2013. Figure 7.3 shows that when fair share is not an issue, then the results for the simulated system and those for the original system actually become identical. Even though there were some differences within the two datasets, this actually represented a 4.6% difference in completion rates within the considered one year time period, as a total of 62376 jobs were completed during 2013. This difference was primarily due to the simulator lacking a fair share component. Taking the results as a whole, it was considered that the simulators deviation from the original data was acceptable. Therefore the simulator was a suitable test for the effectiveness of the *PBStoCondor* code.

Figure 7.1: Comparison of Original and Simulated Data for 2013

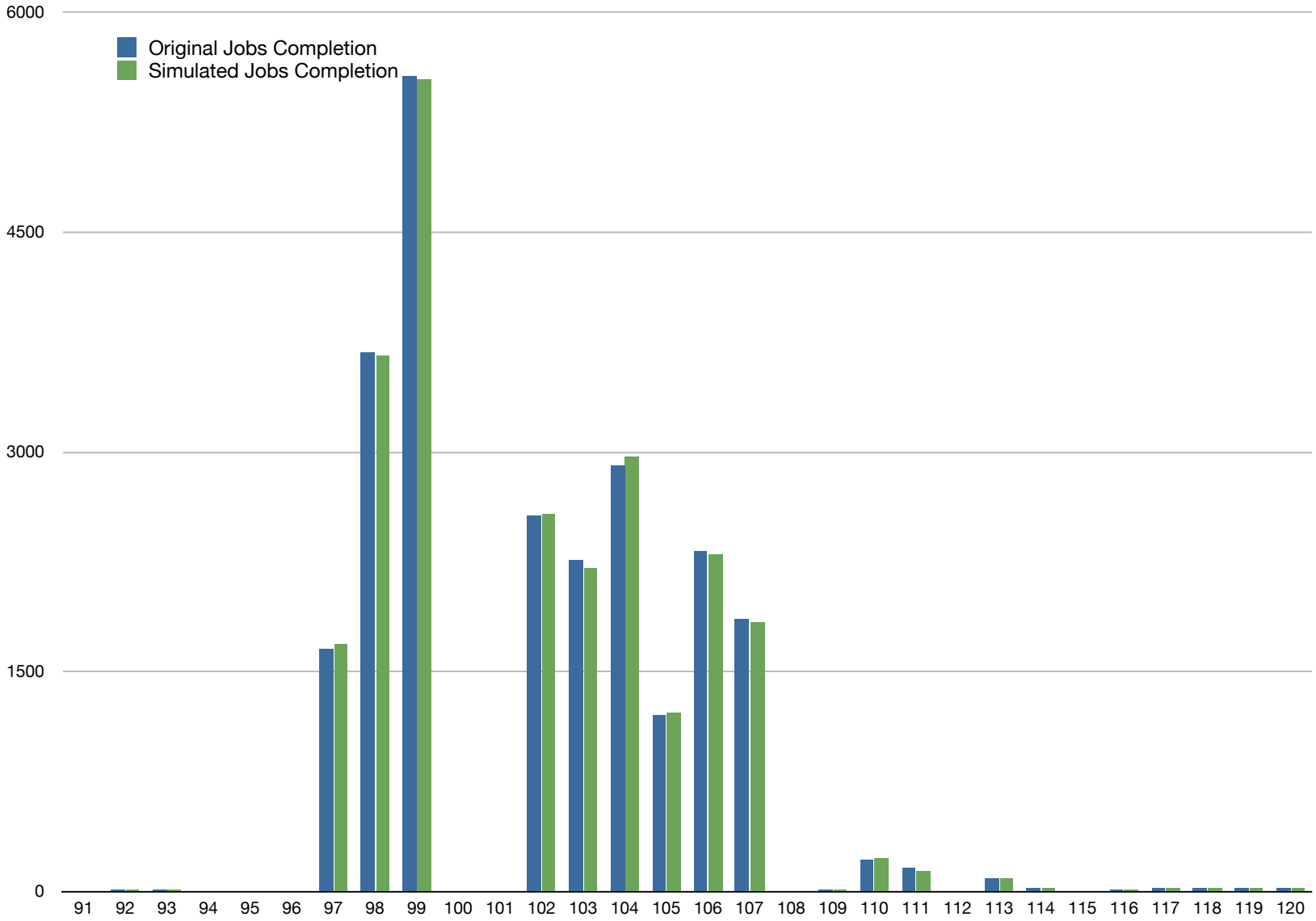

Figure 7.2: Comparison of Original and Simulated Data for April 2013

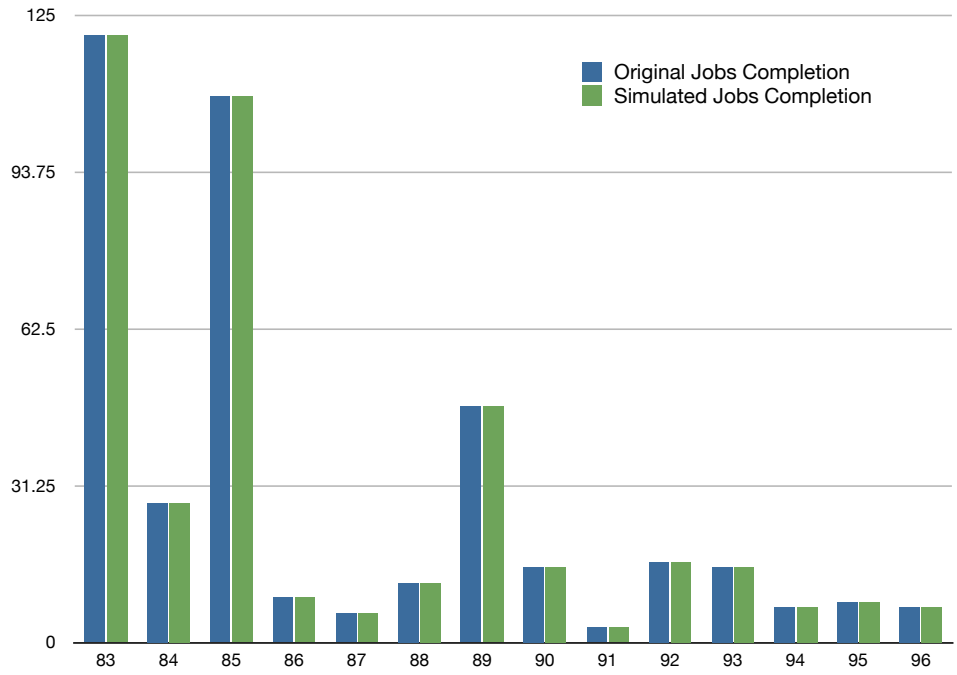

Figure 7.3: Comparison of Original and Simulated Data:
2 Week Period 23/03/2013 - 06/04/2013

7.4 PBStoCondor Results

Using the same procedure, with a system configuration replicating that of a *PBStoCondor* enabled QueensGate Grid (QGG), data for the new system was gathered. The simulator was configured to have 36 quad core nodes to represent the Eridani cluster and 200 single core nodes to represent the available Pool Of Virtual Boxes (POVB) slots within the High Throughput Condor (HTCondor) pool. As with the simulation utilising the Eridani cluster configuration, the logs produced from the *PBStoCondor* simulation were analysed through Hadoop. This provided the job completion rates for the POVB enabled system. Figure 7.4 shows an overview of the 2013 data, comparing the original Torque data and the *PBStoCondor* simulation data. It is clearly shown that the completion rates for the *PBStoCondor* system are higher for a significant portion of the results. When a large number of jobs have been created on the system it appears that the *PBStoCondor* system starts to lag behind the Eridani only system after the initial interval. This however is because *PBStoCondor* is more efficient in job completion and has less jobs left to finish, meaning that the system becomes free more quickly. This will make the system idle and could start saving power sooner than Eridani alone could do. The POVB slots provide additional resources to complete jobs sooner and provide better energy efficiency, since POVB ran on machines with a more energy efficient architecture than the Eridani cluster. Looking at the period covering April 2013 in Figure 7.5 it is clear that the *PBStoCondor* system (Eridani + POVB) consistently out performs the Eridani only system. Lower figures are shown only in in cases where *PBStoCondor* has previously completed a larger number of jobs than the original system. At time interval 111 in Figure 7.5 the *PBStoCondor* system has completed all created jobs while the Eridani system is still processing. This would cause the system to be idle, or in some systems be switched off, for that day,

providing significant savings on overall power usage across a year.

Figure 7.4: Comparison of Original and PBStoCondor Data for 2013

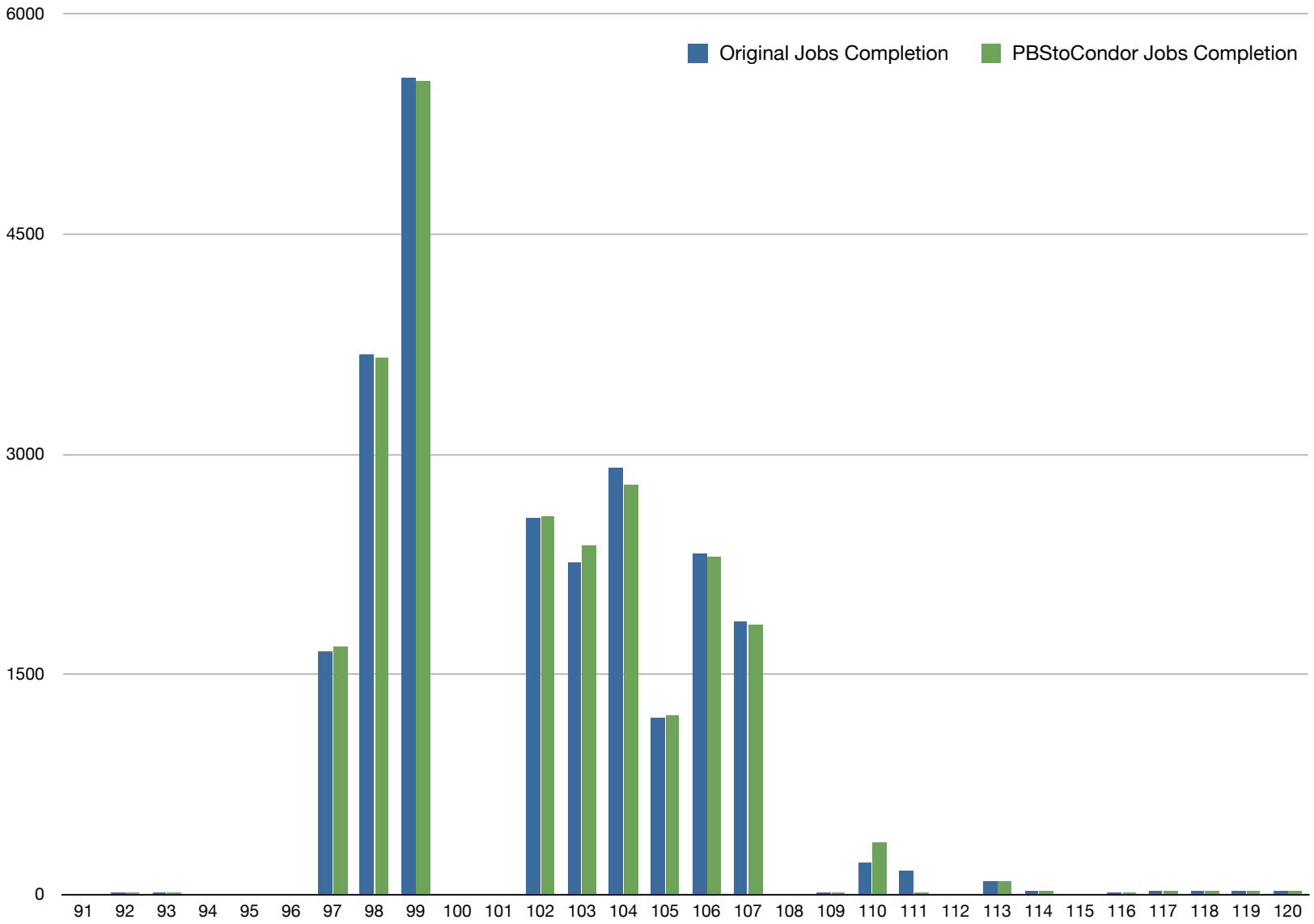

Figure 7.5: Comparison of Original and PBStoCondor Data for April 2013

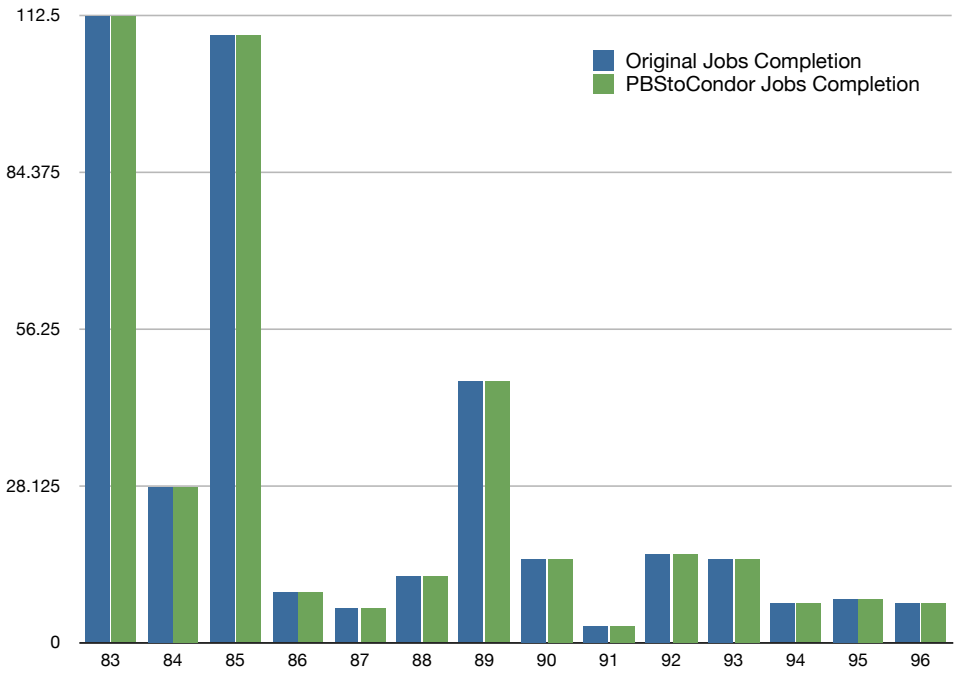

Figure 7.6: Comparison of Original and PBStoCondor Data 2 Week Period (Parallel Load)

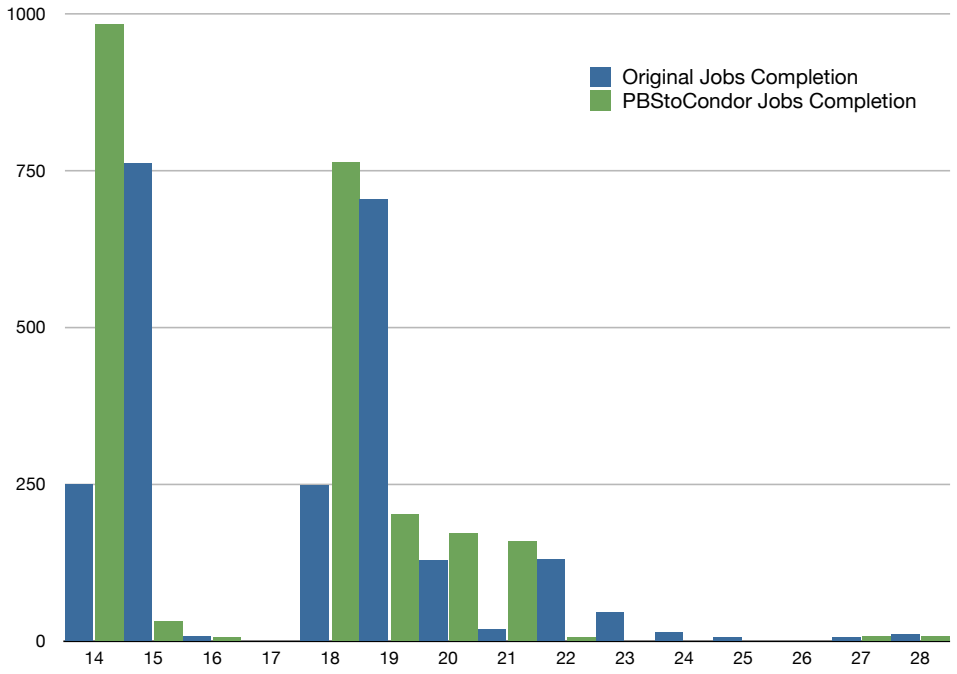

Figure 7.7: Comparison of Original and PBStoCondor Data 2 Week Period (Serial Load)

Figure 7.6 highlights a situation where *PBStoCondor* has no impact on a standard Torque based system. During this period all jobs created on the system were parallel in nature. This presents a time period where *PBStoCondor* is unable to have any impact on system load. Conversely the 2 week period shown on Figure 7.7 depicts a situation where serial jobs make up a significant proportion of submissions. Where this type of situation arises the *PBStoCondor* comes into its own. During this period there are 5 additional days where the system is entirely free. These days could be used to save power as mentioned previously or to process more parallel jobs on the resources that would be made available.

7.5 Summary

Within this section the testing procedure used to evaluate the simulator and *PBStoCondor* system was outlined, followed by consideration of the output generated from the experiments. The first results considered were those which verify the validity of the developed cluster simulator. These results showed that even though the simulator lacks fair-share functionality, it still models the Eridani system very well. The slight deviation in the results as a consequence of this difference only amounted to an overall difference of 4.6%. Therefore any results taken for the *PBStoCondor* enabled system could be considered accurate to $\pm$ 5%. The *PBStoCondor* results showed that for a significant portion of 2013 the developed code would outperform Eridani in terms of job completion. While this system has no impact on periods where many parallel jobs are created, the difference in completion rates is impressive in periods where many serial jobs are created. This improved job completion rate produced multiple days where, had *PBStoConder* been implemented, Eridani would have been entirely idle. This idle time could provide Central Processing Unit (CPU) time for additional jobs or to provide power savings and

reduce the carbon footprint of large parallel compute systems.

Chapter 8

Conclusion

This project has focused on making the best use of institutional computational resources. Current Grid, Cluster, and Cloud software solutions were examined, followed with investigation into how other institutions had configured their available resources. These investigations found a mixture of High Performance Computing (HPC) and High Throughput Computing (HTC) systems with all cross-system campus level jobs originating from a High Throughput Condor (HTCondor) scheduler. Users within the QueensGate Grid (QGG) had displayed a reluctance to move between Torque and HTCondor systems. Administrator observation showed most users remained exclusively on Torque based systems regardless of the type of job being run. This project was undertaken in an attempt to make utilisation of resources more appropriate with minimal user impact. Throughout this project four software frameworks were developed. The initial two of these were *pbsSurge* and *PBStoCondor* followed by the simulator and Hadoop required for evaluation. The aim of these developments was to improve institutional grid user experience, and to facilitate better use of available resources. *pbsSurge* software was developed to allow cloud resources to be utilised within a standard cluster job scheduling environment. This allows very large jobs to be submitted to a system which would not

otherwise have the resources available to service such requests. The developed software was produced as a proof of concept and was tested for pure functionality. However, as QGG users know what is available on a system they tend to remain within those constraints. This means that real quantifiable data was impossible to gather for this system. *pbsSurge* has almost no cost for implementation in terms of processing and memory overheads. If it is implemented on a production system, and the users made aware of its capabilities, it could be used to provide an elastic cluster, able to mould to users requirements. Work in this area made a large contribution to a publication in the International Journal of Advanced Computer Science and Applications (Kureshi et al., 2013). *PBStoCondor* development was focused around better use of existing resources in an attempt to address utilisation issues raised in previous work by the author (Łysik et al., 2013). Within the QGG there is a very large HTCondor pool which was largely under-utilised. The developed software allowed users to maintain use of the more familiar Torque based submission, seamlessly pushing suitable jobs to the HTCondor pool, thus allowing the overall grid to complete computational processes much quicker. This was achieved through better load balancing of the systems. As a consequence this will also potentially provide some power savings on the high power consumption clusters. In order to test *PBStoCondor* further development was required to gain real comparative data. This was developed in the form of a simulator to emulate the system modifications which would have been provided by a *PBStoCondor* deployment. The data produced by this system was then analysed through the final framework developed, using a Hadoop cluster. The results gained showed that the simulator was capable of emulating a real system to within $\pm$4.6%. Even when considering the simulator without a fair share quota system, and assuming that all 200 Pool Of Virtual Boxes (POVB) nodes would always be available, it was demonstrated that the simulator could be used as an effective tool for predicting system behaviour.

The final results obtained from comparison of *PBStoCondor* simulation to historical Eridani logs was very encouraging. These results showed that *PBStoCondor* provided a significant improvement in job completion times over a period of one year. This improvement was particularly apparent when the system had very large numbers of serial jobs submitted. Had this system been deployed at the beginning of the testing period, then there would have been 306 hours, almost 13 days, where the system would have been idle, compared to still working on completing jobs when only Eridani was being used. These periods could represent significant power savings or allow users to submit more jobs, thus making far better use of existing resources.

Chapter 9

Further Work

While a number of aspects have been considered during this project, there are a number of aspects which warrant further investigation. The work completed so far is only suitable for use within a trusted network. This is largely due to a lack of encryption within Torque. Communication between moms is entirely un-encrypted and process transport uses Secure SHell (SSH). The mom communication is entirely inappropriate to be used within an untrusted environment, and it is suspected that the same applies to SSH. To mitigate this, Torque would need to be redesigned in order to incorporate Grid Security Infrastructure (GSI) enabled traffic across all protocols. This approach, while securing communications, would still not be suitable for inter institutional jobs, as Torque would still require a common user file space. This is a problem for which the author currently has no proposed solution. Torque also requires that all possible compute elements are pre-configured, requiring a restart if any additional nodes are to be added. This limits the flexibility of *PBStoCondor* and *pbsSurge*. Further work, allowing Torque to dynamically integrate new resources without requiring a restart, would be greatly beneficial to this project. *pbsSurge* could be improved by adding the ability to use multiple cloud resources, whether private or commercial, along with developing modularity to ex-

tend functionality beyond OpenStack cloud deployments. When considering wider usability *PBStoCondor* is also limited to integration with Torque. Possibly this could be made more modular allowing integration with a variety of schedulers. Many High Throughput Condor (HTCondor) pools are deployed on Windows® based resources. Resource utilisation could be further improved if submission to these types of nodes could be incorporated, since the current solution uses only Pool Of Virtual Boxes (POVB). The simulator developed during the course of this work could also benefit from some further development. From conception, the simulator code was designed to be modular, allowing its behaviour to be easily modified. This could be leveraged in order to incorporate a fair share policy allowing even better representation of the Maui scheduler. Modules could also be created to allow the simulator to simulate a range of schedulers, making it a much more versatile tool.

References

Adaptive Computing. (2014). *Simulation overview.* http://docs.adaptivecomputing.com/maui/16.1simulationoverview.php. Retrieved 2014-04-03, from http://docs.adaptivecomputing.com/maui/16.1simulationoverview.php

Allan, R. (2011). *Infrastructure - northwest grid.* Retrieved 2013-11-15, from http://www.nw-grid.ac.uk/Infrastructure

Beer, D. (2008). *torque for windows.* Retrieved 20/09/2013, from http://www.supercluster.org/pipermail/torqueusers/2012-April/014467.html

Beloglazov, A., Piraghaj, S. F., Alrokayan, M., & Buyya, R. (2012). Deploying openstack on centos using the kvm hypervisor and glusterfs distributed file system. *University of Melbourne*.

Bonner, S., Pulley, C., Kureshi, I., Holmes, V., Brennan, J., & James, Y. (2013, October). Using openstack to improve student experience in an h.e. environment. In *Proceedings of 2013 science and information conference* (pp. 888–893). London, UK: The Science and Information Organization. Retrieved from http://eprints.hud.ac.uk/9895/

Boverhof, J. (2005). *Python wsrf programmersťutorial.* http://acs.lbl.gov/projects/gtg/projects/pyGridWare/doc/tutorial/html/index.html. Retrieved 2014-03-09, from http://acs.lbl.gov/projects/gtg/projects/pyGridWare/doc/tutorial/html/x570.html

Brennan, J., Holmes, V., Kureshi, I., Paprzycki, M., Drozdowicz, M., & Ganzha, M. (2013). Scaling campus grids: Implementing a modified ontology based emi-wms on campus grids. In *The 15th ieee international conference on high performance computing and communications (hpcc 2013).* Retrieved from http://eprints.hud.ac.uk/17337/

Buyya, R., & Murshed, M. (2002a). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, *14*(13-15), 1175–1220.

Buyya, R., & Murshed, M. (2002b). Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, *14*(13-15), 1175–1220. Retrieved from http://dx.doi.org/10.1002/cpe.710 doi: 10.1002/cpe.710

Calleja, M., Beckles, B., Keegan, M., Hayes, M., Parker, A., & Dove, M. T. (2008). Camgrid: Experiences in constructing a university-wide, condor-based grid at the university of cambridge.

CamGrid and PBS university computing service. (2013). Retrieved from http://www.ucs.cam.ac.uk/scientific/camgrid/technical/pbs

Casanova, H. (2001). Simgrid: a toolkit for the simulation of application scheduling. In *Cluster computing and the grid, 2001. proceedings. first ieee/acm international symposium on* (p. 430-437). doi: 10.1109/CCGRID.2001.923223

Casanova, H., Legrand, A., & Quinson, M. (2008). Simgrid: a generic framework for large-scale distributed experiments. In *Proceedings of the tenth international conference on computer modeling and simulation* (pp. 126–131). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dx.doi.org/10.1109/UKSIM.2008.28 doi: 10.1109/UKSIM.2008.28

Chandar, J. (2010). *Join algorithms using map/reduce*. Unpublished master's thesis, University of Edinburgh.

Chang, V., Bacigalupo, D., Wills, G., & De Roure, D. (2010). A categorisation of cloud computing business models. In *Proceedings of the 2010 10th ieee/acm international conference on cluster, cloud and grid computing* (pp. 509–512). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dx.doi.org/10.1109/CCGRID.2010.132 doi: 10.1109/CCGRID.2010.132

Daintith, J., & Wright, E. (2008). *A dictionary of computing*. Retrieved from www.summon.com

Dean, J., & Ghemawat, S. (2004). Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on symposium on opearting systems design & implementation - volume 6* (pp. 10–10). Berkeley, CA, USA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1251254.1251264

Eucalyptus Systems, Inc. (2014). *Eucalyptus* [Page]. Retrieved 2014-02-21, from https://www.eucalyptus.com

European Commission. (2013). *The eu framework programme for research and innovation.* http://ec.europa.eu/programmes/horizon2020/en/what-horizon-2020. Retrieved 2013-11-12, from http://ec.europa.eu/programmes/horizon2020/en/what-horizon-2020

Foster, I., & Kesselman, C. (1999). *The grid: blueprint for a new computing infrastructure* (1st ed.). Morgan Kaufmann Publishers.

Foster, I., Kesselman, C., Nick, J., & Tuecke, S. (2002). The physiology of the grid: An open grid services architecture for distributed systems integration [eb/ol]. *June*, *22*, 24.

Foster, I., Kesselman, C., & Tuecke, S. (2001, mar). *The anatomy of the grid - enabling scalable virtual organizations.* Retrieved 2012-04-18, from http://

arxiv.org/abs/cs/0103025

Foster, I., Kishimoto, H., Savva, A., Berry, D., Grimshaw, A., Maciel, F., ... Von Reich, J. (2005). *The open grid services architecture, version 1.0* (Standard Specification No. 1). OGSA-WG.

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud computing and grid computing 360-degree compared. Retrieved from www.summon.com

Gagliard, F. (2001). *Framework programme 5 grid projects - achievements: Datagrid* (Standard Specification No. 1). CERN. Retrieved from http://webcache.googleusercontent.com/search?q=cache:P3yHbI1muwwJ:ftp://ftp.cordis.europa.eu/pub/ist/docs/grids/datagrid_achievement.pdf+&cd=1&hl=en&ct=clnk&gl=uk&client=firefox-a

Greidanus, P., & Klok, G. (2008, June). Using oscar to win the cluster challenge. In *High performance computing systems and applications, 2008. hpcs 2008. 22nd international symposium on* (p. 47-51). doi: 10.1109/HPCS.2008.22

Herzfeld, D. J., Olson, L. E., & Struble, C. A. (2010). Pools of virtual boxes: Building campus grids with virtual machines. In *Proceedings of the 19th acm international symposium on high performance distributed computing* (pp. 667–675). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1851476.1851575 doi: 10.1145/1851476.1851575

Holmes, V., & Kureshi, I. (2013, April). Creating an he ict infrastructure fit for the 21st century. In *Higher education show 2013.* Retrieved from http://highereducationshow.co.uk/programme/

Jackson, K. (2013). *Openstack cloud computing cookbook*. Packt Publishing. Retrieved from www.summon.com

Kim, D. (2009). *Oscar.* http://svn.oscar.openclustergroup.org/trac/oscar. Retrieved 2012-10-22, from http://svn.oscar.openclustergroup.org/trac/oscar

Konstantinou, I., Floros, E., & Koziris, N. (2012). Public vs private cloud us-

age costs: The stratuslab case. In *Proceedings of the 2nd international workshop on cloud computing platforms* (pp. 3:1–3:6). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2168697.2168700 doi: 10.1145/2168697.2168700

Kranzlmller, D., Lucas, J., & ster, P. (2010). The european grid initiative (egi). In F. Davoli, N. Meyer, R. Pugliese, & S. Zappatore (Eds.), *Remote instrumentation and virtual laboratories* (p. 61-66). Springer US. Retrieved from http://dx.doi.org/10.1007/978-1-4419-5597-5_6 doi: 10.1007/978-1-4419-5597-5_6

Kureshi, I., Pulley, C., Brennan, J., Holmes, V., Bonner, S., & James, Y. (2013, December). Advancing research infrastructure using openstack. *International Journal of Advanced Computer Science and Applications*, *3*(4), 64–70. Retrieved from http://eprints.hud.ac.uk/19421/

Lam, C. (2010). *Hadoop in action*. Manning Publications.

Lee, R. B. (2013). *Security basics for computer architects*. Morgan & Claypool Publishers. Retrieved from www.summon.com

Love, P. (2005). *Beginning unix*. Wrox. Retrieved from www.summon.com

Łysik, K., Wasielewska, K., Paprzycki, M., Drozdowicz, M., Ganzha, M., Brennan, J., ... Kureshi, I. (2013, December). Combining aig agents with unicore grid for improvement of user support. In *2013 first international symposium on computing and networking* (pp. 66–74). Retrieved from http://eprints.hud.ac.uk/19271/

Magoulès, F. (2010). *Fundamentals of grid computing*. CRC Press.

Marowka, A. (2002). What is the grid? *Scalable Computing: Practice and Experience*, *5*(1), 1.

McKinney, W. (2012). *Python for data analysis*. O'Reilly Media. Retrieved from www.summon.com

Mowbray, M. (2007). How web community organisation can help grid computing. *Int. J. Web Based Communities*, *3*(1), 44–54. doi: http://dx.doi.org/10.1504/IJWBC.2007.013773

Naughton, T., Scott, S. L., Barrett, B., Squyres, J., Lumsdaine, A., Fang, Y.-C., & Mashayekhi, V. (2002, November). Looking inside the oscar cluster toolkit. Retrieved 2014-03-12, from http://ftp.dell.com/app/4q02-Osc.pdf

NGS. (2013). *Current members.* http://www.ngs.ac.uk/member-sites/current-members. Retrieved 2013-11-05, from http://web.archive.org/web/20120828100054/http://www.ngs.ac.uk/member-sites/current-members

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., & Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In (p. 124-131). Retrieved from www.summon.com

OpenStack Foundation. (2013a). *Flavors.* http://docs.openstack.org/trunk/openstack-ops/content/flavors.html. Retrieved 2014-04-03, from http://docs.openstack.org/trunk/openstack-ops/content/flavors.html

OpenStack Foundation. (2013b). *Openstack virtual machine image guide.* http://docs.openstack.org/image-guide/content/. Retrieved 2014-04-03, from http://docs.openstack.org/image-guide/content/

Oracle Corporation. (2013). *Oracle grid engine.* Retrieved 20/09/2013, from http://docs.oracle.com/cd/E19080-01/n1.grid.eng6/817-6117/chp1-11/index.html

Platform Computing Corporation. (1996). *Lsf user's guide: Fourth edition.* http://users.cis.fiu.edu/ tho01/psg/3rdParty/lsf4_userGuide/users-title.html. Retrieved 2013-10-22, from http://users.cis.fiu.edu/~tho01/psg/3rdParty/lsf4_userGuide/users-title.html

Proxmox Server Solutions GmbH. (n.d.).

Proxmox Server Solutions GmbH. (2013). *Proxmox virtual environment.* https://www.proxmox.com/proxmox-ve. Retrieved 2013-11-12, from https://www.proxmox.com/proxmox-ve

Quinson, M. (2009, Sept). Simgrid: a generic framework for large-scale distributed experiments. In *Peer-to-peer computing, 2009. p2p '09. ieee ninth international conference on* (p. 95-96). doi: 10.1109/P2P.2009.5284500

Rajaraman, A., & Ullman, J. D. (2012). *Mining of massive datasets*. Cambridge: Cambridge University Press. Retrieved from http://www.amazon.de/Mining-Massive-Datasets-Anand-Rajaraman/dp/1107015359/ref=sr_1_1?ie=UTF8&qid=1350890245&sr=8-1

Reilly, T., Bishop, M., Cast, W., Eskew, P., Farlow, J., Morris, J., ... Tobias, R. (2013). *What is the extreme science and engineering discovery environment* $xsede$ *project.* http://kb.iu.edu/data/baxz.html. Retrieved 2013-11-01, from http://kb.iu.edu/data/baxz.html

Richards, A. (2007, May). The uk national grid service: Grids, ngs and campus grids. In *The 20th open grid forum - ogf20/egee 2nd user forum.* Retrieved from http://www.ogf.org/OGF20/materials/783/ogf20-NGS.pdf

Samuel, C. (2008). *Torque history.* Retrieved 20/09/2013, from http://www.supercluster.org/pipermail/torqueusers/2008-Febuary/006827.html

Steven J. Vaughan-Nichols. (2012). *Canonical switches to openstack for ubuntu linux cloud.* http://www.zdnet.com/blog/open-source/canonical-switches-to-openstack-for-ubuntu-linux-cloud/8875. Retrieved 2014-04-03, from http://www.zdnet.com/blog/open-source/canonical-switches-to-openstack-for-ubuntu-linux-cloud/8875

Thain, D., Tannenbaum, T., & Livny, M. (2005). Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, *17*(24), 323-356. Retrieved from www.summon.com

The OpenStack Foundation. (2013). *Companies supporting the openstack foundation.* http://www.openstack.org/foundation/companies/. Retrieved 2013-11-12, from http://www.openstack.org/foundation/companies/

Univa Corporation. (2013). *history-of-sun-grid-engine.* Retrieved 20/09/2013, from http://www.wheregridenginelives.com/content/history-of-sun-grid-engine

University of Manchester. (2013). *Access to computational grid systems at UoM made easy.* Retrieved from http://talby.rcs.manchester.ac.uk/~rcs/_doc_coll_user/grid_made_easy

University of Reading. (2013). *Campus grid.* https://www.reading.ac.uk/internal/its/e-research/its-eresearch-campusgrid.aspx. Retrieved 2013-11-01, from https://www.reading.ac.uk/internal/its/e-research/its-eresearch-campusgrid.aspx

US Department of Energy. (2013). *Warewulf: Scaleable, modular, adaptable systems management.* http://warewulf.lbl.gov/trac. Retrieved 2013-12-17, from http://warewulf.lbl.gov/trac

Wadia, Y. (2012). The eucalyptus open-source private cloud. *Cloudbook Journal*, *3*(Issue 1). Retrieved 2013-11-15, from http://www.cloudbook.net/resources/stories/the-eucalyptus-open-source-private-cloud

Wallom, D. C. (2007). OxGrid, a campus grid for the university of oxford. In *Proceedings of the UK e-science all hands meeting.* Retrieved from http://www.nesc.ac.uk/talks/ahm2006/625.ppt,Urldate={2013-11-15},Bdsk-Url-1={http://www.nesc.ac.uk/talks/ahm2006/625.ppt}

Wallom, D. C., & Trefethen, A. E. (2006). OxGrid, a campus grid for the university of oxford. In *Proceedings of the UK e-science all hands meeting.* Retrieved 2013-11-15, from http://www.allhands.org.uk/2006/proceedings/papers/625.pdf

White, T. (2010). *Hadoop: The definitive guide* (Second Edition ed.; M. Loukides, Ed.). O'Reilly. Retrieved from http://oreilly.com/catalog/9780596521981

Wilkinson, B. (2010). *Grid computing. techniques and applications*. CRC Press.

World Community Grid. (2013). *About us.* http://www.worldcommunitygrid.org/about_us/viewAboutUs.do. Retrieved 2013-11-04, from http://www.worldcommunitygrid.org/about_us/viewAboutUs.do

Appendix A

Raw Data

A.1 Historic Log Snippet

02/27/2013 00:18:18;E;75153.eridani.qgg.hud.ac.uk;user=u0652238 group=pgr jobname=dlpoly queue=parastd ctime=1361888026 qtime=1361888026 etime=1361888026 start=1361924284 owner=u0652238@qgg.hud.ac.uk exec_host=enode16.eridani.qgg.hud.ac.uk/3+enode16.eridani.qgg.hud.ac.uk/2+enode16.eridani.qgg.hud.ac.uk/1+enode16.eridani.qgg.hud.ac.uk/0+enode11.eridani.qgg.hud.ac.uk/3+enode11.eridani.qgg.hud.ac.uk/2+enode11.eridani.qgg.hud.ac.uk/1+enode11.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List.ncpus=1 Resource_List.neednodes=2:ppn=4 Resource_List.nodect=2 Resource_List.nodes=2:ppn=4 Resource_List.walltime=48:00:00 session=21563 end=1361924298 Exit_status=0 resources_used.cput=00:00:00 resources_used.mem=6536kb resources_used.vmem=189596kb resources_used.walltime=00:00:14

02/27/2013 00:18:22;E;75801.eridani.qgg.hud.ac.uk;user=ngs378 group=ngs jobname=STDIN queue=ngs.ac.uk ctime=1361923036 qtime=1361923036 etime=1361923036 start=1361923036 owner=ngs378@qgg.hud.ac.uk exec_host=enode01.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List.ncpus=1 Resource_List.neednodes=1 Resource_List.nodect=1 Resource_List.nodes=1 Resource_List.walltime=48:00:00 session=4504 end=1361924302 Exit_status=0 resources_used.cput=00:15:25 resources_used.mem=861372kb resources_used.vmem=1023884kb resources_used.walltime=00:21:06

02/27/2013 00:18:24;S;75154.eridani.qgg.hud.ac.uk;user=u0652238 group=pgr jobname=dlpoly queue=parastd ctime=1361888027 qtime=1361888027 etime=1361888027 start=1361924304 owner=u0652238@qgg.hud.ac.uk exec_host=enode16.eridani.qgg.hud.ac.uk/3+enode16.eridani.qgg.hud.ac.uk/2+enode16.eridani.qgg.hud.ac.uk/1+enode16.eridani.qgg.hud.ac.uk/0+enode11.eridani.qgg.hud.ac.uk/3+enode11.eridani.qgg.hud.ac.uk/2+enode11.eridani.qgg.hud.ac.uk/1+enode11.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00

Resource_List.ncpus=1 Resource_List.neednodes=2:ppn=4 Resource_List.nodect=2
Resource_List.nodes=2:ppn=4 Resource_List.walltime=48:00:00
02/27/2013 00:18:39;E;75154.eridani.qgg.hud.ac.uk;user=u0652238 group=pgr jobname=dlpoly
queue=parastd ctime=1361888027 qtime=1361888027 etime=1361888027 start=1361924304
owner=u0652238@qgg.hud.ac.uk exec_host=enode16.eridani.qgg.hud.ac.uk/3+enode16.eridani
.qgg.hud.ac.uk/2+enode16.eridani.qgg.hud.ac.uk/1+enode16.eridani.qgg.hud.ac.uk/0+
enode11.eridani.qgg.hud.ac.uk/3+enode11.eridani.qgg.hud.ac.uk/2+enode11.eridani.qgg.
hud.ac.uk/1+enode11.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00
Resource_List.ncpus=1 Resource_List.neednodes=2:ppn=4 Resource_List.nodect=2
Resource_List.nodes=2:ppn=4 Resource_List.walltime=48:00:00 session=21650 end
=1361924319 Exit_status=0 resources_used.cput=00:00:00 resources_used.mem=808kb
resources_used.vmem=13352kb resources_used.walltime=00:00:15
02/27/2013 00:18:45;Q;75812.eridani.qgg.hud.ac.uk;queue=ngs.ac.uk
02/27/2013 00:18:45;Q;75813.eridani.qgg.hud.ac.uk;queue=ngs.ac.uk
02/27/2013 00:18:45;S;75812.eridani.qgg.hud.ac.uk;user=ngs378 group=ngs jobname=STDIN
queue=ngs.ac.uk ctime=1361924325 qtime=1361924325 etime=1361924325 start=1361924325
owner=ngs378@qgg.hud.ac.uk exec_host=enode01.eridani.qgg.hud.ac.uk/0 Resource_List.
cput=10000:00:00 Resource_List.ncpus=1 Resource_List.neednodes=1 Resource_List.nodect
=1 Resource_List.nodes=1 Resource_List.walltime=48:00:00
02/27/2013 00:18:45;S;75813.eridani.qgg.hud.ac.uk;user=ngs378 group=ngs jobname=STDIN
queue=ngs.ac.uk ctime=1361924325 qtime=1361924325 etime=1361924325 start=1361924325
owner=ngs378@qgg.hud.ac.uk exec_host=enode01.eridani.qgg.hud.ac.uk/2 Resource_List.
cput=10000:00:00 Resource_List.ncpus=1 Resource_List.neednodes=1 Resource_List.nodect
=1 Resource_List.nodes=1 Resource_List.walltime=48:00:00
02/27/2013 00:18:49;S;75155.eridani.qgg.hud.ac.uk;user=u0652238 group=pgr jobname=dlpoly
queue=parastd ctime=1361888027 qtime=1361888027 etime=1361888027 start=1361924329
owner=u0652238@qgg.hud.ac.uk exec_host=enode16.eridani.qgg.hud.ac.uk/3+enode16.eridani
.qgg.hud.ac.uk/2+enode16.eridani.qgg.hud.ac.uk/1+enode16.eridani.qgg.hud.ac.uk/0+
enode11.eridani.qgg.hud.ac.uk/3+enode11.eridani.qgg.hud.ac.uk/2+enode11.eridani.qgg.
hud.ac.uk/1+enode11.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00
Resource_List.ncpus=1 Resource_List.neednodes=2:ppn=4 Resource_List.nodect=2
Resource_List.nodes=2:ppn=4 Resource_List.walltime=48:00:00

A.2 Simulator Log Snippet

```
26/02/2013 18:15:29;E;75153.0;user=u0652238 queue=parastd ctime=1361888026 qtime
    =1361902515 etime=1361888026 start=1361902515 end=1361902529 Resource_List.nodes=2:ppn
    =4 duration=14
26/02/2013 18:15:44;E;75154.0;user=u0652238 queue=parastd ctime=1361888027 qtime
    =1361902529 etime=1361888027 start=1361902529 end=1361902544 Resource_List.nodes=2:ppn
    =4 duration=15
26/02/2013 18:15:55;E;75155.0;user=u0652238 queue=parastd ctime=1361888027 qtime
    =1361902544 etime=1361888027 start=1361902544 end=1361902555 Resource_List.nodes=2:ppn
    =4 duration=11
26/02/2013 18:16:09;E;75156.0;user=u0652238 queue=parastd ctime=1361888033 qtime
    =1361902555 etime=1361888034 start=1361902555 end=1361902569 Resource_List.nodes=2:ppn
    =4 duration=14
26/02/2013 18:16:34;E;75157.0;user=u0652238 queue=parastd ctime=1361888036 qtime
    =1361902569 etime=1361888036 start=1361902569 end=1361902594 Resource_List.nodes=2:ppn
    =4 duration=25
26/02/2013 18:16:58;E;75158.0;user=u0652238 queue=parastd ctime=1361888037 qtime
    =1361902594 etime=1361888037 start=1361902594 end=1361902618 Resource_List.nodes=2:ppn
    =4 duration=24
26/02/2013 18:17:34;E;75159.0;user=u0652238 queue=parastd ctime=1361888038 qtime
    =1361902618 etime=1361888038 start=1361902618 end=1361902654 Resource_List.nodes=2:ppn
    =4 duration=36
26/02/2013 18:17:56;E;75160.0;user=u0652238 queue=parastd ctime=1361888042 qtime
    =1361902654 etime=1361888042 start=1361902654 end=1361902676 Resource_List.nodes=2:ppn
    =4 duration=22
26/02/2013 18:18:21;E;75161.0;user=u0652238 queue=parastd ctime=1361888043 qtime
    =1361902676 etime=1361888043 start=1361902676 end=1361902701 Resource_List.nodes=2:ppn
    =4 duration=25
26/02/2013 18:18:58;E;75162.0;user=u0652238 queue=parastd ctime=1361888049 qtime
    =1361902701 etime=1361888049 start=1361902701 end=1361902738 Resource_List.nodes=2:ppn
    =4 duration=37
26/02/2013 18:19:27;E;75163.0;user=u0652238 queue=parastd ctime=1361888050 qtime
    =1361902738 etime=1361888050 start=1361902738 end=1361902767 Resource_List.nodes=2:ppn
    =4 duration=29
26/02/2013 18:19:48;E;75164.0;user=u0652238 queue=parastd ctime=1361888051 qtime
    =1361902767 etime=1361888051 start=1361902767 end=1361902788 Resource_List.nodes=2:ppn
    =4 duration=21
26/02/2013 18:20:07;E;75165.0;user=u0652238 queue=parastd ctime=1361888052 qtime
    =1361902788 etime=1361888052 start=1361902788 end=1361902807 Resource_List.nodes=2:ppn
    =4 duration=19
```

```
26/02/2013 18:20:31;E;75166.0;user=u0652238 queue=parastd ctime=1361888053 qtime
    =1361902807 etime=1361888053 start=1361902807 end=1361902831 Resource_List.nodes=2:ppn
    =4 duration=24
26/02/2013 18:21:01;E;75167.0;user=u0652238 queue=parastd ctime=1361888053 qtime
    =1361902831 etime=1361888053 start=1361902831 end=1361902861 Resource_List.nodes=2:ppn
    =4 duration=30
26/02/2013 18:21:33;E;75168.0;user=u0652238 queue=parastd ctime=1361888055 qtime
    =1361902861 etime=1361888056 start=1361902861 end=1361902893 Resource_List.nodes=2:ppn
    =4 duration=32
26/02/2013 18:22:06;E;75169.0;user=u0652238 queue=parastd ctime=1361888058 qtime
    =1361902893 etime=1361888058 start=1361902893 end=1361902926 Resource_List.nodes=2:ppn
    =4 duration=33
```

Appendix B

PBStoCondor Translator Code

```
#!/bin/bash
#
# Generic PBS to Condor run script v1.0
#
# John Brennan
#
# Change Log
# Version       Date          Description
#-------------+-------------+---------------------+
# 1.0         | 18Jun13     |    Initial release  |
#-------------+-------------+---------------------+

# This script is designed to run under a pbs_mom on a condor
# headnode. It will take any arguements passed to PBS, generate
# a condor jdl to run the relevant /usr/ngs script on the compute
# node with the required arguements and then continue to watch
# the condor job until it completes.

# Source condor files #
#source /opt/condor-7.6.7/condor.sh # Required for old qggcondor deployment

# Variable Creation #
JOBSCRIPT="$PBS_O_WORKDIR/$PBS_JOBNAME.jcl"
ERRFILE="$PBS_O_WORKDIR/$PBS_JOBNAME.err"
OUTFILE="$PBS_O_WORKDIR/$PBS_JOBNAME.out"
LOGFILE="$PBS_O_WORKDIR/$PBS_JOBNAME.log"
```

```
# Sort and format file syntax, fixing any non standard paths that may have been used
for i in $@
        do test -f $i && RAWINP="$RAWINP $i"; done
INPUTFILES=`echo $RAWINP | sed "s# #, #g"`
for i in $@
        do
        if [ ! -f $i ];
        then ARGS="$ARGS $i"
        else ARGS="$ARGS `basename $i`"
        fi;
done
# Strip path from executable name
EXE=`basename $0`

# Create submission script #
echo "universe = vanilla" > $JOBSCRIPT
echo 'requirements = (OpSys == "LINUX" && Arch == "X86_64")' >> $JOBSCRIPT
echo "transfer_input_files = $INPUTFILES" >> $JOBSCRIPT
echo "executable = /usr/ngs/$EXE" >> $JOBSCRIPT
echo "transfer_executable = false" >> $JOBSCRIPT
echo "arguments = $ARGS" >> $JOBSCRIPT
echo "error = $ERRFILE" >> $JOBSCRIPT
echo "log = $LOGFILE" >> $JOBSCRIPT
echo "output = $OUTFILE" >> $JOBSCRIPT
echo "should_transfer_files = YES" >> $JOBSCRIPT # Required for new condor deployment
echo "when_to_transfer_output=ON_EXIT_OR_EVICT" >> $JOBSCRIPT # Required for new condor
    deployment
#echo "transfer_files = ALWAYS" >> $JOBSCRIPT # Required for old qggcondor deployment
echo "initial_dir= $PBS_O_WORKDIR" >> $JOBSCRIPT
echo "Notification = ERROR" >> $JOBSCRIPT
echo "queue 1" >> $JOBSCRIPT

# Submit job to queue and determine job number#
JOBNO=`condor_submit $JOBSCRIPT | tail -1 | cut -c 31- | sed s/\.$//`

# Wait for job to finish #
condor_wait $LOGFILE $JOBNO > /dev/null

# Push all output to pbs_mom
echo "------------------------------------------"
echo "Begin Logging"
```

```
echo "-------------------------------------------------"
cat $LOGFILE
echo "-------------------------------------------------"
echo "End Logging"
echo "-------------------------------------------------"
echo "-------------------------------------------------"
echo "Begin Output"
echo "-------------------------------------------------"
cat $OUTFILE
echo "-------------------------------------------------"
echo "End Output"
echo "-------------------------------------------------"
echo "-------------------------------------------------" >&2
echo "Begin Error" >&2
echo "-------------------------------------------------" >&2
cat $ERRFILE >&2
echo "-------------------------------------------------" >&2
echo "End Error" >&2
echo "-------------------------------------------------" >&2

# Fix Condor logging to show PBStoCondor has been used
LINE=`grep -rne "ClusterId = $JOBNO" /var/lib/condor/spool/history | head -n 1 | sed s
    /:.*//`
if [[ -z "$INPUTFILES" ]]
   then DIFF=29
   else DIFF=30
fi
LINE=`expr $LINE - $DIFF`
STRING="'$LINE s/.*/Owner = "$USER-PtoC"/' /var/lib/condor/spool/history"
STRING=$(echo $STRING | sed 's/Owner = /Owner = \"/')
STRING=$(echo $STRING | sed 's/PtoC/PtoC\"/')
eval sed -i -c $STRING

# Clean up #
rm -rf $JOBSCRIPT $LOGFILE $OUTFILE $ERRFILE

exit $?
$
```

Appendix C

PBStoCondor-Daemon Code

C.1 Main

```
#!/usr/bin/python
import sys
from subprocess import Popen, PIPE

# Setup Logging
import logging
logger = logging.getLogger('PBStoCondord')
hdlr = logging.FileHandler('/var/log/PbstoCondord.log')
formatter = logging.Formatter('%(asctime)s %(name)s %(levelname)s %(message)s', datefmt='%
    b %d %Y %H:%M:%S')
hdlr.setFormatter(formatter)
logger.addHandler(hdlr)
logger.setLevel(logging.ERROR)
logger.propagate = False

# Define the Struct class. This creates a structure to be used within an array.
# Populating it with the "Headings" hostname(string), active(bool), running(bool),
# mport(int), rport(int).
class Struct:
    def __init__(self, hostname):
     self.hostnames = hostname
     active = False
```

```
        running = False
        mport = 0
        rport = 0

# Function to read configuration information
def sysinfoParser():
# Set config file path and read mode
  try:
    config = file('/etc/pbstocondord/pbstocondord.config', 'r')
  except IOError as (errno, strerror):
            logger.critical("IOError({0}): {1}".format(errno, strerror))
            logger.critical("Config file can not be found. Exiting")
            sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
            sys.exit(1)
# Define global variables which can be used outside this function
  global node_prefix
  node_prefix = ""
  global max_moms_int
  global min_port_int
  global max_port_int
# Define local variables, These need to be predefined as empty for the config file
# error checking to function properly
  max_moms = ""
  min_port = ""
  max_port = ""
# Split config lines using = as delimiter, and assign values to variables
# Ignoring any lines starting with # or whitespace
  for line in config:
    if line.startswith('#') or line.startswith(' '):
      pass
    else:
      temp = line.split('=')
          if temp[0] == "NODE_PREFIX":
              node_prefix = temp[1].rstrip('\n')
            elif temp[0] == 'MAX_MOMS':
                  max_moms = temp[1].rstrip('\n')
            elif temp[0] == 'MIN_PORT':
                  min_port = temp[1].rstrip('\n')
            elif temp[0] == 'MAX_PORT':
                  max_port = temp[1].rstrip('\n')
            elif temp[0] == 'DEBUG_LEVEL':
```

```
            debug_level = str(temp[1].rstrip('\n'))
# Check for missing values in config file
  if not node_prefix or not max_moms or not min_port or not max_port:
    logger.critical('Missing variables in config file, Please check your config file for
        errors')
    sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
    sys.exit(1)
  else:
    pass
  if not debug_level:
    pass
  elif debug_level == "DEBUG":
    logger.setLevel(logging.DEBUG)
    logger.debug("setting debug level to DEBUG")
  elif debug_level == "INFO":
    logger.setLevel(logging.INFO)
    logger.info("setting debug level to INFO")
  elif debug_level == "WARNING":
    logger.setLevel(logging.WARNING)
    logger.warning("setting debug level to WARNING")
  elif debug_level == "ERROR":
    logger.setLevel(logging.ERROR)
    logger.error("setting debug level to ERROR")
  elif debug_level == "CRITICAL":
    logger.setLevel(logging.CRITICAL)
    logger.critical("setting debug level to CRITICAL")

  max_moms_int = int(max_moms)
  min_port_int = int(min_port)
  max_port_int = int(max_port)
  logger.info("NODE_PREFIX = " + str(node_prefix))
  logger.info("MAX_MOMS = " + str(max_moms))
  logger.info("MIN_PORT = " + str(min_port))
  logger.info("MAX_PORT = " + str(max_port))


# Function to enumerate all required permutations of configuration information and store
    them
# in a structured array
def valenumerator():
```

```
# Needed to avoid "UnboundLocalError: local variable 'min_port_int' referenced before
    assignment"
  global min_port_int
  global mom_counter
  global mom_table
# Define an array called mom_table
  mom_table =[]
# Initialise counter as a string
  counter = "1"
  mom_counter = 0
  for i in range(0,max_moms_int):
# Catch errors in config pertaining to allowed port range
    if min_port_int >= max_port_int:
      logger.critical('Not enough ports assigned for given number of moms. Please ensure
          total port range is equal to double the number of required moms!')
      sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
      sys.exit(1)
    else:
      pass
    hosttemp = "".join((node_prefix, counter))
    counter = int(counter)
    counter = counter + 1
    counter = str(counter)
    logger.debug("Hostname = " + str(hosttemp))
    if i == 0:
      mport = min_port_int
      logger.debug("mom_service_port = " + str(mport))
      min_port_int = min_port_int + 1
      rport = min_port_int
      logger.debug("mom_manager_port = " + str(rport))
    else:
      min_port_int = min_port_int + 1
      mport = min_port_int
      logger.debug("mom_service_port = " + str(mport))
      min_port_int = min_port_int + 1
      rport = min_port_int
      logger.debug("mom_manager_port = " + str(rport))

    mom_table.append(Struct(hosttemp))
    mom_table[mom_counter].mport = mport
    mom_table[mom_counter].rport = rport
```

```
    mom_table[mom_counter].active = False
    mom_table[mom_counter].running = False
    mom_counter = mom_counter + 1

# Function to find the number of POVB nodes currently avaliable
def povbgetnum():
  global conodes
  try:
    pipe0 = Popen(["condor_status"], stdout=PIPE)
  except OSError as (errno, strerror):
              logger.critical("OSError({0}): {1}".format(errno, strerror))
              logger.critical("condor_status can not be found. Exiting")
              sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
    sys.exit(1)
  pipe1 = Popen(["grep", "/LINUX"], stdin=pipe0.stdout, stdout=PIPE)
  pipe2 = Popen(["cut", "-c", "24-26"], stdin=pipe1.stdout, stdout=PIPE)
  try:
    totconodes = int(pipe2.communicate()[0].rstrip('\n'))
  except ValueError:
    logger.debug("condor_status returned NULL, setting totconodes to 0")#debug
    totconodes = 0
    logger.debug("totconodes = " + str(totconodes))#debug

  try:
    pipe00 = Popen(["condor_status"], stdout=PIPE)
  except OSError as (errno, strerror):
              logger.critical("OSError({0}): {1}".format(errno, strerror))
              logger.critical("condor_status can not be found. Exiting")
              sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
    sys.exit(1)
  pipe01 = Popen(["grep", "/LINUX"], stdin=pipe00.stdout, stdout=PIPE)
  pipe02 = Popen(["cut", "-c", "30-32"], stdin=pipe01.stdout, stdout=PIPE)
  try:
    ownconodes = int(pipe02.communicate()[0].rstrip('\n'))
  except ValueError:
    logger.debug("condor_status returned NULL, setting ownconodes to 0")#debug
    ownconodes = 0
    logger.debug("ownconodes = " + str(ownconodes))#debug

  logger.info("totconodes = " + str(totconodes))
  logger.info("ownconodes = " + str(ownconodes))
```

```
    conodes = totconodes - ownconodes
    logger.info("conodes = " + str(conodes))

# Function to find active moms
def momgetactive():
    global momactive_int
    global mom_table
    temp = ""
    momactive_int = 0
    try:
        pipe0 = Popen(["pbsnodes", "-l", "active"], stdout=PIPE)
    except OSError as (errno, strerror):
                logger.critical("OSError({0}): {1}".format(errno, strerror))
                logger.critical("pbsnodes can not be found. Exiting..")
                sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
        sys.exit(1)
    pipe1 = Popen(["grep", str(node_prefix)], stdin=pipe0.stdout, stdout=PIPE)
    sedstr = "sed -e 's/ *job-exclusive//g'"
    pipe2 = Popen([sedstr], stdin=pipe1.stdout, stdout=PIPE, shell=True)
    momactive = pipe2.communicate()[0]
    logger.info("active moms = " + str(momactive))
# Parse momactive to find an int value of how many moms are running and update mom_table
    with those that are running
    for line in momactive:
        temp = temp + line
        if line == '\n':
            momactive_int = momactive_int + 1
            temp = int(temp.lstrip(node_prefix).rstrip('\n')) - 1
            mom_table[temp].active = True
            logger.debug(temp + " has been marked as active = " + mom_table[temp].active)
            temp = ""
        else:
            pass
    logger.info("active moms int = " + str(momactive_int))

# Find moms marked as running in mom_table, or in first pass check with pbs_server
def momgetrunning():
    global momrunning_int
    global firstpass
    momrunning_int = 0
    temp = ""
```

```
  if firstpass == True:
    firstpass = False
    logger.info("Entering first pass of momgetrunning()")
    try:
      pipe0 = Popen(["pbsnodes", "-l", "free"], stdout=PIPE)
    except OSError as (errno, strerror):
      logger.critical("OSError({0}): {1}".format(errno, strerror))
                      logger.critical("pbsnodes can not be found. Exiting")
      sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
                      sys.exit(1)
    pipe1 = Popen(["grep", str(node_prefix)], stdin=pipe0.stdout, stdout=PIPE)
    sedstr = "sed -e 's/ *free//g'"
    pipe2 = Popen([sedstr], stdin=pipe1.stdout, stdout=PIPE, shell=True)
    momrunning = pipe2.communicate()[0]
    logger.info("running moms = " + str(momrunning).rstrip('\n'))
    for line in momrunning:
      temp = temp + line
      if line == '\n':
        momrunning_int = momrunning_int + 1
        temp = int(temp.lstrip(node_prefix).rstrip('\n')) - 1
        mom_table[temp].running = True
        logger.debug(str(temp) + " has been marked as running = " + str(mom_table[temp].
            running))
        temp = ""
  else:
    for i in range(0,mom_counter):
      if mom_table[i].running == True:
        momrunning_int = momrunning_int + 1
      else:
        pass
  logger.info("mom running int = " + str(momrunning_int))

# If there are more condor nodes than moms start more moms
def startmoms():
  global mom_table
  if momrunning_int < conodes:
    logging.info("momrunning int < conodes = True")
    for i in range(0,mom_counter):
      if momrunning_int < conodes:
        logger.debug("startmoms momrunning int = " + str(momrunning_int))
        logger.debug("startmoms conodes = " + str(conodes))
```

```
            if mom_table[i].running == False:
              command = "pbs_mom -m -M " + str(mom_table[i].mport) + " -R " + str(mom_table[i
                  ].rport) + " -H " + str(mom_table[i].hostnames)
              logger.info("Staring a mom, with the following command:" + command)
              try:
                p = Popen(command, stdout=PIPE, shell=True)
              except OSError as (errno, strerror):
                                  logger.critical("OSError({0}): {1}".format(errno, strerror))
                                  logger.critical("pbs_mom can not be found. Exiting")
                                  sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
                sys.exit(1)
              result = p.communicate()[0]
              logging.debug("Start mom Popen vars are : " + str(vars(p)))
              logging.debug("Result of start mom is: " + str(result))
              mom_table[i].running = True
              momgetrunning()
            else:
              pass
          else:
            pass
        else:
          pass

# If there are less condor nodes than moms stop some moms, ensuring moms to be stopped are
    not active
def killmoms():
  global mom_table
  if momrunning_int > conodes:
    logger.info("momrunning_int > conodes = True")
    for i in range(0,mom_counter):
      if momrunning_int > conodes:
        if mom_table[i].active == False and mom_table[i].running == True:
          command = "momctl -s -p " + str(mom_table[i].rport)
          logger.info("Stopping a mom with the following command: " + command)
          try:
            p = Popen(command, stdout=PIPE, shell=True)
          except OSError as (errno, strerror):
                                  logger.critical("OSError({0}): {1}".format(errno, strerror))
                                  logger.critical("momctl can not be found. Exiting")
                                  sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
            sys.exit(1)
```

```
            result = p.communicate()[0]
            logging.debug("Stop mom Popen vars are : " + str(vars(p)))
                                        logging.debug("Result of stop mom is: " + str(
                                            result))
            mom_table[i].running = False
            momgetrunning()
          else:
            pass
        else:
          pass
    else:
      pass

def go():
    import time
    global firstpass
    firstpass = True
    sysinfoParser()
    logger.info("Initialising ......")
    valenumerator()
    logger.info("Entering main loop ......")

    while(1):
      momgetrunning()
      momgetactive()
      povbgetnum()
      startmoms()
      povbgetnum()
      killmoms()
      time.sleep(30)
```

C.2 init.py

```
#!/usr/bin/python

import sys, time
from pbstocondord_daemon import Daemon
from pbstocondord_main import go

class MyDaemon(Daemon):
        def run(self):
                while True:
                        go()

if __name__ == "__main__":
        daemon = MyDaemon('/tmp/pbstocondord.pid')
        if len(sys.argv) == 2:
                if 'start' == sys.argv[1]:
                        daemon.start()
                elif 'stop' == sys.argv[1]:
                        daemon.stop()
                elif 'restart' == sys.argv[1]:
                        daemon.restart()
                elif 'status' == sys.argv[1]:
                        daemon.status()
                else:
                        print "Unknown command"

                        sys.exit(2)
                sys.exit(0)
        else:
                print "usage: %s start|stop|restart" % sys.argv[0]
                sys.exit(2)
```

C.3 RHEL init

```
#!/bin/sh
# Startup script for PBStoCondor Daemon
# chkconfig: 2345 55 25
# processname: pbstocondord
# config: /etc/pbstocondord.config

### BEGIN INIT INFO
# Provides: pbstocondord
# Required-Start: $local_fs $network $syslog
# Required-Stop: $local_fs $syslog
# Should-Start: $syslog
# Should-Stop: $network $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start up the PBStoCondor server daemon
# Description:
#
### END INIT INFO

# If the daemon is not there, then exit.
test -x $EXEC || exit 5
EXEC=/opt/pbstocondord/pbstocondord_init
case "$1" in
  start)
    $EXEC $1;;
        stop)
                $EXEC $1;;
        restart)
                $EXEC $1;;
        status)
                $EXEC $1;;
  *)
    # For invalid arguments, print the usage message.
      echo "Usage: $0 {start|stop|restart|status}"
      exit 2;;
esac
{\color{white} $}
```

C.4 Config

```
# PBStoCondord config file

NODE_PREFIX=conode
MAX_MOMS=200
MIN_PORT=20000
MAX_PORT=30000
DEBUG_LEVEL=INFO
```

Appendix D

pbsSurge Code

D.1 Submitter

```
#!/usr/bin/python
# Standard imports
import os, time, uuid, sys, re, fileinput, shutil
from subprocess import Popen, PIPE
# Credentials functions
import credentials
# Import nova API
import novaclient.v1_1.client as nvclient
# Import keystone/glance API
import keystoneclient.v2_0.client as ksclient
#import glanceclient.v2.client as glclient
import glanceclient.v1.client as glclient

# Get creds and vars
nvcreds = credentials.get_nova_creds()
nova = nvclient.Client(**nvcreds)
servid = uuid.uuid4().hex
flavid = uuid.uuid4().hex
torqimg ="CentOS-6"
sshkey ="tauceti_root"
ippool ="pbsSurge"
rawmem="0gb"
flavmem="1024"
```

```
flavhdd="10"
flavcpu="1"
jobscript = sys.argv[1]
shutil.copy(jobscript, 'tmp')
jobscript = file(jobscript, 'r')

for line in jobscript:
                if line.startswith('#PBS'):
                        temp = line.split(' ')
                        if temp[1] == "-l":
        temp1 = temp[2].split(':')
        for var in temp1:
            temp2 = var.split('=')
            if temp2[0] == 'ncpus':
              flavcpu = temp2[1].rstrip('\n')
            elif temp2[0] == "mem":
              rawmem = temp2[1].rstrip('\n')
            elif temp2[0] == "ppn":
                                                    flavcpu = temp2[1].rstrip('\n')
            elif temp2[0] == "nodes":
                                                    numnodes = temp2[1].rstrip('\n')
r = re.compile("([0-9]+)([a-zA-Z]+)")
m = r.match(rawmem)
if m.group(2) == 'gb' or m.group(2) == 'GB':
  flavmem = int(m.group(1))*1024
elif m.group(2) == 'kb' or m.group(2) == 'KB':
  flavmem = int(m.group(1))/1024


#print "num nodes in script " + numnodes
#print "num cpus in script " + flavcpu
#print "mem  in script " + str(flavmem)

if int(flavmem) < 1048 and int(flavcpu) < 4:
## Run job as normal
  print "/usr/bin/qsub" + " " + sys.argv[1]
  os.remove('tmp')
  sys.exit(0)
else:
## Modify submit queue in job script
  for  line in fileinput.FileInput('tmp', inplace=1):
```

```
    if line.startswith('#PBS -q'):
#     line = re.sub(r'#PBS -q.*', r'#PBS -q highmemq', line)
      line = re.sub(r'#PBS -q.*', r'#PBS -q serialstd', line)
      print line.rstrip('\n')
    else:
      print line.rstrip('\n')
## Create Flavour
  reqflavor = nova.flavors.create(name=flavid, ram=flavmem, disk=flavhdd, vcpus=flavcpu)

## Start instance
  image = nova.images.find(name=torqimg)
  instance = nova.servers.create(name=servid, image=image, flavor=reqflavor, key_name=
      sshkey)
# print instance
# print str(instance)
# Poll at 5 second intervals, until the status is no longer 'BUILD'
  status = instance.status
  while status == 'BUILD':
      time.sleep(5)
    # Retrieve the instance again so the status field updates
      instance = nova.servers.get(instance.id)
      status = instance.status
# print "status: %s" % status
# Attach floating IP
  ip = nova.floating_ips.create(pool=ippool)
  instance.add_floating_ip(ip)
# print "floating_ip= " + ip
## Run job
  pipe0 = Popen(["/usr/bin/qsub", "tmp"], stdout=PIPE, stderr=PIPE)
  jobno, joberr = pipe0.communicate()
  if joberr.startswith('qsub: submit error'):
    jobno = '0'
    print str(joberr).rstrip('\n')
    sys.exit(1)
  else:
    pass
  os.remove('tmp')
  print str(jobno).rstrip('\n')
  pipe1 = Popen(["/home/u0765098/testing/pbsSurge/watcher.py", jobno, servid, flavid, "&"]
      , stdout=PIPE)
  sys.exit(0)
```

D.2 Watcher

```
#!/usr/bin/python
import os, time, sys
from subprocess import Popen, PIPE
# Credentials functions
import credentials
# Import nova API
import novaclient.v1_1.client as nvclient

## sys args:
jobno = sys.argv[1]
servid = sys.argv[2]
flavid = sys.argv[3]
jobstat = "1"
nvcreds = credentials.get_nova_creds()
nova = nvclient.Client(**nvcreds)
instance = nova.servers.find(name=servid)
status = instance.status
instid = instance.id
reqflavor = nova.flavors.find(name=flavid)
ip = nova.floating_ips.find(instance_id=instid)

## Wait for job to finish
while jobstat:
  pipe1 = Popen(["/usr/bin/qstat", jobno], stdout=PIPE, stderr=PIPE)
        jobstat = pipe1.communicate()[0]
        time.sleep(30)

## Kill instance
instance.delete()
while status == 'Deleting':
  time.sleep(5)

## Delete Flavor
nova.flavors.delete(reqflavor)
nova.floating_ips.delete(ip)
sys.exit(0)
```

Appendix E

Simulator Code

E.1 resources.txt snippet

```
## Scheduler Simulator resource configuration file
0 4
1 4
2 4
3 4
4 4
5 4
6 1
7 1
8 1
9 1
```

E.2 simulator.py

```
#!/usr/bin/python
import sys, time

# Setup Logging
import logging
logger = logging.getLogger('resource_parser')
hdlr = logging.FileHandler('resource_parser.log')
formatter = logging.Formatter('%(asctime)s %(name)s %(levelname)s %(message)s', datefmt='%
    b %d %Y %H:%M:%S')
```

```
hdlr.setFormatter(formatter)
logger.addHandler(hdlr)
DEBUG2=0
DEBUG3=0
if DEBUG2:
  logger.setLevel(logging.DEBUG)
else:
  logger.setLevel(logging.ERROR)
logger.propagate = False
global running_table
running_table = []
global queueing_table
queueing_table = []
global sorting_table
sorting_table = []
global previous_trigger

#################
## Function to read the resources file and populate the resource_table[] based on the
    number of cores avaliable.
## The resourec file consists of space seperated values node number and number of no_cpus
def resource_parse():
  global resource_table
  resource_table = []

  try:
    resource_file = file('resources.txt', 'r')
  except IOError as (errno, strerror):
              logger.critical("IOError({0}): {1}".format(errno, strerror))
              logger.critical("resources.txt can not be found. Exiting")
              sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
              sys.exit(1)

  for line in resource_file:
## Ensure line starts with a numerical digit before processing
    if line[0].isdigit():
      temp = line.split(' ')
      resource_table.append([])

      for i in range(0,int(temp[1])):
        resource_table[int(temp[0])].append(1)
```

```
    else:
      pass

  #for j in range(0,int(temp[0])+int(1)):
  # print resource_table[j]
  #print resource_table[1][0]

  return resource_table #FIXME(return is not needed)
  #FIXME resource_table is alredy global

##################
# Define the Struct0 class. This creates a structure to be used within an array.
# Populating it with Headings
class Struct0:
   def __init__(self, jobno):
    self.jobnos = jobno
    ctime = 0
          nodes = 0
          fjobno = ""
          ppn = 0
          queue = ""
          username = ""
          qtime = 0
          etime = 0
          start = 0
          end = 0
          duration = 0
          resources = []

##################
# Define the Struct1 class. This creates a structure to be used within an array.
# Populating it with Headings
class Struct1:
   def __init__(self, trigger_time):
    self.trigger_times = trigger_time
    jobnos = ""
      # Resource_List.nodes=2:ppn=4
          run_OR_sort = ""

##################
def log_parse(torque_log):
```

```
  global init_table
  init_table = []
  global init_tab_length
  init_tab_length = 0
## The following values only curremtly remain global to give the other tables some initial
    values.
  #global jobno
  #global nodes
  #global ppn
  #global queue
  #global ctime
  #global qtime
  #global etime
  #global start
  #global end
  #global username
  #global duration

  try:
    log_file = file(torque_log, 'r')
  except IOError as (errno, strerror):
    logger.critical("IOError({0}): {1}".format(errno, strerror))
    logger.critical("accounting log can not be found. Exiting")
    sys.stderr.write("!!! ERROR !!!\nPlease Check Logs\n")
    sys.exit(1)

  logger.debug("Log Parse: Populating the init_table")
  for line in log_file:
    temp0 = line.split(";")
    #print temp0
    #print temp0[1]
## Only process E records from accounting log
    if temp0[1] == "E":
      fjobno = str(temp0[2])
      temp1 = fjobno.split(".")
      if "[" not in temp1[0]:
        jobno = temp1[0]+".0"
      else:
        k=temp1[0].split("[")
        j=k[1].split("]")
        jobno=k[0]+"."+j[0]
```

```
    #jobno=k[0]
  temp1 = temp0[3].split(" ")
  #print temp1
  for i in range(0, len(temp1)):
    #print temp1[i]
    if temp1[i].startswith("user"):
      username = str(temp1[i].split("=")[1])
    elif temp1[i].startswith("queue"):
      queue = str(temp1[i].split("=")[1])
    elif temp1[i].startswith("ctime"):
      ctime = str(temp1[i].split("=")[1])
    elif temp1[i].startswith("qtime"):
      qtime = str(temp1[i].split("=")[1])
    elif temp1[i].startswith("etime"):
      etime = str(temp1[i].split("=")[1])
    elif temp1[i].startswith("start"):
      start = str(temp1[i].split("=")[1])
    elif temp1[i].startswith("end"):
      end = str(temp1[i].split("=")[1])
    elif temp1[i].startswith("Resource_List.nodes"):
      temp2 = temp1[i].split(":")
      nodes = temp2[0].split("=")[1]
      if len(temp2) == 2:
        ppn = int(temp2[1].split("=")[1])
      else:
        ppn = 1

  duration = int(end) - int(start)

  if find_space(int(nodes),int(ppn),jobno):
    init_table.append(Struct0(jobno))
    init_table[init_tab_length].fjobno = fjobno
    init_table[init_tab_length].queue = queue
    init_table[init_tab_length].ctime = int(ctime)
    init_table[init_tab_length].qtime = int(qtime)
    init_table[init_tab_length].etime = int(etime)
    init_table[init_tab_length].start = int(start)
    init_table[init_tab_length].end = int(end)
    init_table[init_tab_length].nodes = int(nodes)
    init_table[init_tab_length].ppn = int(ppn)
    init_table[init_tab_length].username = username
```

```
        init_table[init_tab_length].duration = int(duration)
        init_tab_length = init_tab_length + 1
      else:
        logger.debug("Log Parse: System Configuration is not compatible for job: "+str(
            jobno))

  logger.debug("Log Parse: Completed the init_table")

##################
def bubble_sort(seq,index):
    #Inefficiently sort the mutable sequence (list) in place.
    #   seq MUST BE A MUTABLE SEQUENCE.
  mycount=1
    #   As with list.sort() and random.shuffle this does NOT return
    # 9-12-13 rosettacode.org/wiki/Sorting_algorithms/Bubble_sort#Python
  if index == 'init':
    changed = True
    while changed:
      print mycount
      changed = False
      for i in range(len(seq) - 1):
        if float(seq[i].jobnos) > float(seq[i+1].jobnos):
          seq[i], seq[i+1] = seq[i+1], seq[i]
          changed = True
      mycount += 1
  elif index == 'sort':
    changed = True
    while changed:
      changed = False
      for i in range(len(seq) - 1):
        if seq[i].trigger_times > seq[i+1].trigger_times:
          seq[i], seq[i+1] = seq[i+1], seq[i]
          changed = True

  return None

##################
def make_busy(nodes,ppn,jobnos):

  no_cpus = 0
  no_nodes = 0
```

```
  pass1 = 0
  cont = True
  v = ""
  global made_busy
  made_busy = []

  logger.debug("Making Busy for "+str(jobnos))
  for i in range(0,len(resource_table)):
    if no_nodes != int(nodes):
      if len(resource_table[i]) == ppn and cont == True:
        for j in range(0,len(resource_table[i])):
          if resource_table[i][j] == 1:
            no_cpus = no_cpus + 1
            if no_cpus == ppn:
              for k in range(0,no_cpus):
                resource_table[i][k] = 0
                made_busy.extend((i,k))
              no_nodes = no_nodes + 1
              no_cpus = 0
#             pass1 = 1
            else:
              logger.debug("Busy-Pass 1: the number of cpus has already been allocated")
              pass
          else:
            logger.debug("Busy-Pass 1: the core is busy")
            cont = False
      else:
        logger.debug("Busy-Pass 1: nodes size does not match ppn exactly")
        pass
    else:
      logger.debug("Busy-Pass 1: node allocations are now complete")
      pass

# if pass1 == 0:
  for i in range(0,len(resource_table)):
    if no_nodes != int(nodes):
      if len(resource_table[i]) >= ppn:
        core_sum=0
        for j in range(0,len(resource_table[i])):
          core_sum=core_sum + resource_table[i][j]
        #print core_sum, ppn
```

```
        if core_sum >= ppn:
          no_nodes=no_nodes+1
          no_cpus=0
          for j in range(0,len(resource_table[i])):
            if resource_table[i][j] == 1 and no_cpus<ppn:
              no_cpus = no_cpus + 1
              resource_table[i][j] = 0
              made_busy.extend((i,j))
            else:
              logger.debug("Busy-Pass 2: core is either busy or all allocations complete")
              pass
        else:
          logger.debug("Busy-Pass 2: node does not have enough free cores")
          pass
      else:
        logger.debug("Busy-Pass 2: node does not have enough cores to match ppn")
        pass
    else:
      logger.debug("Busy-Pass 2: node allocations are now complete")
      pass

  if no_nodes != int(nodes):
    return 0
  else:
    print "Making Allocation: ",nodes,ppn
    if 1:
      for xx in range(0,len(resource_table)):
        print resource_table[xx]
      print "----------------------------------------------------------------------"
    return made_busy

##################
def make_free(busy_cores):

  for i in range(0,len(busy_cores),2):
    resource_table[busy_cores[i]][busy_cores[i+1]] = 1

##################
def find_space(nodes,ppn,jobnos):

  no_cpus = 0
```

```
no_nodes = 0
pass1 = 0
cont = True
node_sum=0
core_sum=0
v = ""

logger.debug("Find Space for "+str(jobnos))
# Identify if we even have enough nodes to make such an allocation
for i in range(0,len(resource_table)):
  if len(resource_table[i]) >= ppn:
    node_sum=node_sum+1
if node_sum>=nodes:

  #Pass 1- find those nodes that match exactly
  for i in range(0,len(resource_table)):
    cont=True
    if no_nodes != int(nodes):
      if len(resource_table[i]) == ppn:
        for j in range(0,len(resource_table[i])):
          if resource_table[i][j] == 1 and cont == True:
            no_cpus = no_cpus + 1
            if no_cpus == ppn:
              no_nodes = no_nodes + 1
              no_cpus = 0
            else:
              logger.debug("Space-Pass 1: not enough space on the node")
              pass
          else:
            logger.debug("Space-Pass 1: the core is busy")
            cont = False
            no_cpus=0
      else:
        logger.debug("Space-Pass 1: nodes size does not match ppn exactly")
        pass
    else:
      logger.debug("Space-Pass 1: node allocations are now complete")
      pass

  for i in range(0,len(resource_table)):
    if no_nodes != int(nodes):
```

```
        if len(resource_table[i]) > ppn:
          core_sum=0
          for j in range(0,len(resource_table[i])):
            core_sum=core_sum + resource_table[i][j]
          #print core_sum, ppn
          if core_sum >= ppn:
            no_nodes=no_nodes+1
            no_cpus=0
            for j in range(0,len(resource_table[i])):
                if resource_table[i][j] == 1 and no_cpus<ppn:
                  no_cpus = no_cpus + 1
                else:
                  logger.debug("Space-Pass 2: core is either busy or all allocations
                      complete")
                  pass
          else:
            logger.debug("Space-Pass 2: node does not have enough free cores")
            pass
        else:
          logger.debug("Space-Pass 2: node does not have enough cores to match ppn")
          pass
      else:
        logger.debug("Space-Pass 2: node allocations are now complete")
        pass

  if no_nodes != int(nodes):
    return 0
  else:
    return 1

##################
def print_logs(completed_job):

  tstamp = time.strftime('%d/%m/%Y %H:%M:%S', time.localtime(completed_job.end))
  new_log = open(str(sys.argv[1]) + ".new", 'a')
  new_log.write(tstamp+";E;"+completed_job.jobnos+";user=" + completed_job.username +"
      queue="+ completed_job.queue+" ctime="+str(completed_job.ctime)+" qtime="+str(
      completed_job.qtime)+" etime="+str(completed_job.etime)+" start="+str(completed_job.
      start)+" end="+str(completed_job.end)+" Resource_List.nodes="+str(completed_job.
      nodes)+":ppn="+str(completed_job.ppn)+" duration="+str(completed_job.duration)+"\n")
  new_log.close()
```

```
##################
def move_job(src_tab,src_index,dest_tab,resources,sort_flag):

  dest_rec_index = ""

  if sort_flag == 'R':
    dest_tab.append(Struct1(src_tab[src_index].start + src_tab[src_index].duration))
    dest_tab[len(dest_tab)-1].jobnos = src_tab[src_index].jobnos
    dest_tab[len(dest_tab)-1].run_OR_sort = sort_flag
    dest_rec_index = len(dest_tab)-1
  elif sort_flag == 'Q':
    dest_tab.append(Struct1(src_tab[src_index].ctime))
    dest_tab[len(dest_tab)-1].jobnos = src_tab[src_index].jobnos
    dest_tab[len(dest_tab)-1].run_OR_sort = sort_flag
    dest_rec_index = len(dest_tab)-1

  else:
    dest_tab.append(Struct0(src_tab[src_index].jobnos))
    dest_tab[len(dest_tab)-1].nodes = src_tab[src_index].nodes
    dest_tab[len(dest_tab)-1].ppn = src_tab[src_index].ppn
    dest_tab[len(dest_tab)-1].queue = src_tab[src_index].queue
    dest_tab[len(dest_tab)-1].username = src_tab[src_index].username
    dest_tab[len(dest_tab)-1].ctime = src_tab[src_index].ctime
    dest_tab[len(dest_tab)-1].qtime = src_tab[src_index].qtime
    dest_tab[len(dest_tab)-1].etime = src_tab[src_index].etime
    dest_tab[len(dest_tab)-1].start = src_tab[src_index].start
    dest_tab[len(dest_tab)-1].end = src_tab[src_index].end
    dest_tab[len(dest_tab)-1].duration = src_tab[src_index].duration
    if resources == -999:
      dest_tab[len(dest_tab)-1].resources = 0
    else:
      dest_tab[len(dest_tab)-1].resources = resources
    dest_rec_index = len(dest_tab)-1

  return dest_rec_index

##################
def find_index(jobno,table):

  for i in range(0,len(table)):
```

```
    if table[i].jobnos == jobno:
      return i

##################
def make_running(i,table,previous_trigger):

  print "Allocating job  : ",table[i].jobnos,table[i].nodes,table[i].ppn
  if 0:
    for xx in range(0,len(resource_table)):
      print resource_table[xx]
    print "------------------------------------------------------------------"
  running_job_index=move_job(table,i,running_table,make_busy(table[i].nodes,table[i].ppn,
      table[i].jobnos),-999)

  #print "Trigger Time: ",previous_trigger
  if previous_trigger != 0:
    if previous_trigger<running_table[running_job_index].ctime:
      if running_table[running_job_index].etime <= running_table[running_job_index].ctime:
        running_table[running_job_index].start=running_table[running_job_index].ctime
      else:
        running_table[running_job_index].start=running_table[running_job_index].etime
    else:
      running_table[running_job_index].start=previous_trigger
  else:
    if running_table[running_job_index].etime <= running_table[running_job_index].ctime:
      running_table[running_job_index].start=running_table[running_job_index].ctime
    else:
      running_table[running_job_index].start=running_table[running_job_index].etime

  running_table[running_job_index].qtime=running_table[running_job_index].start
  running_table[running_job_index].end=running_table[running_job_index].start+
      running_table[running_job_index].duration
  #print str(running_table[running_job_index].end)
  #move_job(table,i,sorting_table,-999,'R')
  #bubble_sort(sorting_table,'sort')
  if DEBUG2:
    display_table("running_table")
    display_table("queueing_table")
    display_table("sorting_table")

  return running_job_index
```

```
##################
def make_queued(i):

  print "Can not make allocation for :",init_table[i].jobnos,init_table[i].nodes,
      init_table[i].ppn
  print "Job being queued: ",init_table[i].jobnos
  queued_job_index=move_job(init_table,i,queueing_table,-999,-999)
  #move_job(init_table,i,sorting_table,-999,'Q')
  #bubble_sort(sorting_table,'sort')
  if DEBUG2:
    display_table("running_table")
    display_table("queueing_table")
    display_table("sorting_table")
  return queued_job_index

##################
def process_queue_items(init_trigger):
  global previous_trigger
  j=0
  #for j in range(0,len(sorting_table)):
  while j < len(sorting_table):
    if init_trigger > sorting_table[j].trigger_times:
      if sorting_table[j].run_OR_sort == 'R':

        make_free(running_table[find_index(sorting_table[j].jobnos,running_table)].
            resources)
        completed_job=running_table.pop(find_index(sorting_table[j].jobnos,running_table))
        print_logs(completed_job)
        previous_trigger = sorting_table[j].trigger_times

        print "Popping: " + sorting_table[j].jobnos
        print "_____________________________________________________________"
        sorting_table.pop(j)

        if DEBUG3:
          print "Calling inside while (poped jobs)"
          display_table("sorting_table")

        j=0
        #break
```

```
      else:
        queue_index=find_index(sorting_table[j].jobnos,queueing_table)
        if find_space(queueing_table[queue_index].nodes,queueing_table[queue_index].ppn,
            queueing_table[queue_index].jobnos):
          print "Setting a Queued job with index: " + str(queue_index) + " and jobid: " +
              sorting_table[j].jobnos + " to running\n"
          running_job_index=make_running(queue_index,queueing_table,previous_trigger)

          sorting_table[j].trigger_times=queueing_table[queue_index].duration+
              previous_trigger
          sorting_table[j].run_OR_sort = 'R'
          bubble_sort(sorting_table,'sort')

          queueing_table.pop(queue_index)

          j=0
          #break
        else:
          j += 1

      if len(sorting_table) == 0:
        #print "for: the length is now 0"
        break
    else:
      j += 1

  return 0

##################
def display_table(table):

  if table == "sorting_table":
    print "\n-------------------------sorting-------------------------------"
    print "Trigger      | Job No   |   Queue"
    for k in range(0,len(sorting_table)):
      print str(sorting_table[k].trigger_times) + "     " + sorting_table[k].jobnos + "
          " + sorting_table[k].run_OR_sort
    print "---------------------------------------------------------------"
  elif table == "queueing_table":
    print "\n-------------------------queueing-------------------------------"
    print "Job No | ctime       | nodes | ppn | duration"
```

```
    for k in range(0,len(queueing_table)):
      print queueing_table[k].jobnos + "    " + str(queueing_table[k].ctime) + "    " + str(
          queueing_table[k].nodes) + "        " + str(queueing_table[k].ppn) + "        " +
          str(queueing_table[k].duration)
    print "-----------------------------------------------------------------"
  else:
    print "\n-------------------------running-------------------------------"
    print "Job No | ctime       | nodes | ppn | duration | start | end | resources "
    for k in range(0,len(running_table)):
      print running_table[k].jobnos + "    " + str(running_table[k].ctime) + "    " + str(
          running_table[k].nodes) + "        " + str(running_table[k].ppn) + "        " +
          str(running_table[k].duration) + "        " + str(running_table[k].start) + "
              " + str(running_table[k].end) + "        " + str(running_table[k].resources
          )
    print "-----------------------------------------------------------------"

##################
if __name__ == "__main__":

  resource_parse()
  log_parse(sys.argv[1])
  debug_showdata=0
  previous_trigger = 0

  bubble_sort(init_table,'init')

  if debug_showdata:
    for j in range(0,len(resource_table)):
      print resource_table[j]

  for i in range(0,init_tab_length):
    print "\n-----------------------------------------------------------"
    print "New Job is: " + init_table[i].jobnos + " at time " + str(init_table[i].etime)
    print "-----------------------------------------------------------"
    if len(sorting_table) == 0:
      running_job_index=make_running(i,init_table,0)
      move_job(running_table,running_job_index,sorting_table,-999,'R')
      bubble_sort(sorting_table,'sort')
    else:
      if init_table[i].ctime < sorting_table[0].trigger_times:
        if find_space(init_table[i].nodes,init_table[i].ppn,init_table[i].jobnos):
```

```
            running_job_index=make_running(i,init_table,0)

            move_job(running_table,running_job_index,sorting_table,-999,'R')
            bubble_sort(sorting_table,'sort')

          else:
            queued_job_index=make_queued(i)

            move_job(queueing_table,queued_job_index,sorting_table,-999,'Q')
            bubble_sort(sorting_table,'sort')
        else:
          if DEBUG3:
            print "Calling before the while"
            display_table("sorting_table")
          done_sorting=1
          while (sorting_table[0].trigger_times < init_table[i].ctime) and (done_sorting ==
              1):
            #print "this"
            done_sorting=process_queue_items(init_table[i].ctime)
            #print "this2"
            if len(sorting_table) == 0:
              #print "while: the length is now 0"
              break

          #mytemp=find_space(init_table[i].nodes,init_table[i].ppn,init_table[i].jobnos)
          #print mytemp
          if find_space(init_table[i].nodes,init_table[i].ppn,init_table[i].jobnos):
            running_job_index=make_running(i,init_table,previous_trigger)
            move_job(running_table,running_job_index,sorting_table,-999,'R')
            bubble_sort(sorting_table,'sort')
          else:
            queued_job_index=make_queued(i)
            move_job(queueing_table,queued_job_index,sorting_table,-999,'Q')
            bubble_sort(sorting_table,'sort')

######
if DEBUG3:
  print "Calling before the while"
  display_table("sorting_table")
while len(sorting_table) > 0:
  process_queue_items(9999999999)
```

```
    if len(sorting_table) == 0:
      #print "while: the length is now 0"
      break

###########
  print "\n\n"
  if 0:
    for i in range(0,init_tab_length):
      print "\n-----------------------------------------------------------------------"
      print "jobno is - " + init_table[i].jobnos
      print "queue is - " + init_table[i].queue
      print "ctime is - " + str(init_table[i].ctime)
      print "qtime is - " + str(init_table[i].qtime)
      print "etime is - " + str(init_table[i].etime)
      print "start is - " + str(init_table[i].start)
      print "end is   - " + str(init_table[i].end)
      print "nodes is - " + str(init_table[i].nodes)
      print "ppn is   - " + str(init_table[i].ppn)
      print "duration - " + str(init_table[i].duration)
      print "-----------------------------------------------------------------------"

  if 1:
    print "\n----------------------closing statement----------------------"
    print "Jobs processed: ",str(len(init_table))
    print "-------------------------------------------------------------------"

    display_table("running_table")

    display_table("queueing_table")

    display_table("sorting_table")


    print "\n-------------------------------system-----------------------------"
    for i in range(0,len(resource_table)):
      print resource_table[i]
    print "-------------------------------------------------------------------"
```

Appendix F

Hadoop Code

F.1 Driver

F.1.1 DriverPBSLog.java

```
package Driver;
import java.net.URI;
import java.util.Date;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.lib.MultipleTextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.conf.Configuration;
@SuppressWarnings("unused")
public class DriverPBSLog
{
  static String path_prefix = "/LUBM/";
  static String logsPath = "_logs/";
```

```
static String ISWCPath = null;
static String tmpPath = null;
static String factsPath = null;
static Integer numOfMaps = 1, numOfReduces = 1;
private static void parseArgs(String[] args){
  try {
    String arg;
      int i = 0;
      while (i < args.length) {
        arg = args[i++];
          if (arg.equals("-maps")) {
            if (i < args.length) {
              arg = args[i++];
              numOfMaps = Integer.parseInt(arg);
              if (numOfMaps < 1)
                throw new NumberFormatException();
            }
            else
              throw new NumberFormatException();
          }
          else if (arg.equals("-reduces")) {
            if (i < args.length) {
              arg = args[i++];
              numOfReduces = Integer.parseInt(arg);
              if (numOfReduces < 1)
                throw new NumberFormatException();
              }
            else
              throw new NumberFormatException();
          }
          else if (arg.equals("-path_prefix")) {
            if (i < args.length) {
              arg = args[i++];
              path_prefix = arg;
            }
          }
          else if (arg.equals("-facts_path")) {
            if (i < args.length) {
              arg = args[i++];
              factsPath = arg;
            }
```

```
            }
        }
    }catch (Exception e) {
        System.err.println(
            "Usage: RunExperiments\n" +
            "\t[-maps <num of maps (1~" + Integer.MAX_VALUE + ")>]\n" +
                "\t[-reduces <num of reduces (0~" + Integer.MAX_VALUE +")>]\n" +
                "\t[-path_prefix <prefix for all paths>]\n" +
                "\t[-facts_path <path of the facts>]\n"
        );
        System.exit(0);
    }
  }
public static void main(String[] args) throws Exception
 {
  Configuration conf = new Configuration();
  // Need to add these config files so that it looks for hdfs and not local file system
  conf.addResource(new Path("/usr/local/hadoop/hadoop-1.0.3/conf/core-site.xml"));
  conf.addResource(new Path("/usr/local/hadoop/hadoop-1.0.3/conf/hdfs-site.xml"));
  Job job = new Job(conf);
  job.setJobName("Pass 1");
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  job.setMapperClass(Mappers.Pass1.class);
  job.setReducerClass(Reducers.Reduce1.class);
  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);
  String input = "input/";
  String output = "output/pass1/";
  FileInputFormat.setInputPaths(job, new Path(input));
  FileOutputFormat.setOutputPath(job, new Path(output));
  //job.setNumReduceTasks(0);
  job.setJarByClass(DriverPBSLog.class);
  Date startTime = new Date();
  System.out.println("Job Started: " + startTime);
  int exitCode = job.waitForCompletion(true) ? 0 : 1;
  if(exitCode == 0)
  {
    Date end_time = new Date();
    System.out.println("Job Ended: " + end_time);
```

```
    System.out.println("Pass 1 Took " + (end_time.getTime() - startTime.getTime()) /1000.0
        + " seconds.");
  }
  else
  {
    System.out.println("Job Failed - Please Check Console Output");
  }
  // second pass looking for completion rate
  conf.addResource(new Path("/usr/local/hadoop/hadoop-1.0.3/conf/core-site.xml"));
  conf.addResource(new Path("/usr/local/hadoop/hadoop-1.0.3/conf/hdfs-site.xml"));
  job = new Job(conf);
  job.setJobName("Pass 2");
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  job.setMapperClass(Mappers.Pass2.class);
  job.setReducerClass(Reducers.Reduce1.class);
  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);
  output = "output/pass2/";
  FileInputFormat.setInputPaths(job, new Path(input));
  FileOutputFormat.setOutputPath(job, new Path(output));
  //job.setNumReduceTasks(0);
  job.setJarByClass(DriverPBSLog.class);
  startTime = new Date();
  System.out.println("Job Started: " + startTime);
  exitCode = job.waitForCompletion(true) ? 0 : 1;
  if(exitCode == 0)
  {
    Date end_time = new Date();
    System.out.println("Job Ended: " + end_time);
    System.out.println("Pass 2 Took " + (end_time.getTime() - startTime.getTime()) /1000.0
        + " seconds.");
  }
  else
  {
    System.out.println("Job Failed - Please Check Console Output");
  }
}
```

F.2 Mappers

F.2.1 Pass1.java

```
package Mappers;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class Pass1 extends Mapper<LongWritable, Text, Text, IntWritable>
{
  private static int startEpochTime = 1293840000;
  private static int interval = 86400;
  private final static IntWritable one = new IntWritable(1);
  public void map(LongWritable key, Text value, Context context) throws IOException,
      InterruptedException
  {
    Text timeInterval = new Text();
    String linevalue = value.toString();
    String lineValSplitOnComma[] = linevalue.split(";");
    if(lineValSplitOnComma[1].equals("S"))
    {
      String lineValSplitOnWhiteSpace[] = lineValSplitOnComma[3].split(" ");
      for(int i = 0; i < lineValSplitOnWhiteSpace.length; i++)
      {
        if(lineValSplitOnWhiteSpace[i].startsWith("ctime="))
        {
          String ctimeSplit[] = lineValSplitOnWhiteSpace[i].split("=");
          int ctimeEpoch = Integer.parseInt(ctimeSplit[1]);
          int differance = (int)(ctimeEpoch-startEpochTime)/interval;
          timeInterval.set(new Text(String.valueOf(differance)));
          context.write(timeInterval, one);
        }
      }
    }
}
}
```

F.2.2 Pass2.java

```
package Mappers;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class Pass2 extends Mapper<LongWritable, Text, Text, IntWritable>
{
  private static int startEpochTime = 1293840000;
  private static int interval = 86400;
  private final static IntWritable one = new IntWritable(1);
  public void map(LongWritable key, Text value, Context context) throws IOException,
      InterruptedException
  {
    Text timeInterval = new Text();
    String linevalue = value.toString();
    String lineValSplitOnComma[] = linevalue.split(";");
    if(lineValSplitOnComma[1].equals("E"))
    {
      String lineValSplitOnWhiteSpace[] = lineValSplitOnComma[3].split(" ");

      for(int i = 0; i < lineValSplitOnWhiteSpace.length; i++)
      {
        if(lineValSplitOnWhiteSpace[i].startsWith("end="))
        {
          String endtimeSplit[] = lineValSplitOnWhiteSpace[i].split("=");
          int endtimeEpoch = Integer.parseInt(endtimeSplit[1]);
          int differance = (int)(endtimeEpoch-startEpochTime)/interval;
          timeInterval.set(new Text(String.valueOf(differance)));
          context.write(timeInterval, one);
        }
      }
    }
}
}
```

F.2.3 uEcomprate.java

```
package Mappers;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class uEcomprate extends Mapper<LongWritable, Text, Text, IntWritable>
{
  private static int startEpochTime = 1362268800;
  private static int interval = 900;
  private final static IntWritable one = new IntWritable(1);
  public void map(LongWritable key, Text value, Context context) throws IOException,
      InterruptedException
  {
    String userName = null;
    String timeInterval = null;
    String linevalue = value.toString();
    String lineValSplitOnComma[] = linevalue.split(";");
    if(lineValSplitOnComma[1].equals("E"))
    {
      String lineValSplitOnWhiteSpace[] = lineValSplitOnComma[3].split(" ");
      for(int i = 0; i < lineValSplitOnWhiteSpace.length; i++)
      {
        if(lineValSplitOnWhiteSpace[i].startsWith("user="))
        {
          String userSplit[] = lineValSplitOnWhiteSpace[i].split("=");
          userName = userSplit[1];
          //
          for(int j = 0; j < lineValSplitOnWhiteSpace.length; j++)
          {
            if(lineValSplitOnWhiteSpace[j].startsWith("end="))
            {
              String ctimeSplit[] = lineValSplitOnWhiteSpace[j].split("=");
              int ctimeEpoch = Integer.parseInt(ctimeSplit[1]);
              int differance = (int)(ctimeEpoch-startEpochTime)/interval;
              timeInterval = String.valueOf(differance);
              context.write(new Text(userName+"-"+timeInterval), one);
            }
}}}}}}
```

F.2.4 userExtractor.java

```
package Mappers;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class userExtractor extends Mapper<LongWritable, Text, Text, IntWritable>
{
  private static int startEpochTime = 1362268800;
  private static int interval = 900;
  private final static IntWritable one = new IntWritable(1);
  public void map(LongWritable key, Text value, Context context) throws IOException,
      InterruptedException
  {
    String userName = null;
    String timeInterval = null;
    String linevalue = value.toString();
    String lineValSplitOnComma[] = linevalue.split(";");
    if(lineValSplitOnComma[1].equals("S"))
    {
      String lineValSplitOnWhiteSpace[] = lineValSplitOnComma[3].split(" ");
      for(int i = 0; i < lineValSplitOnWhiteSpace.length; i++)
      {
        if(lineValSplitOnWhiteSpace[i].startsWith("user="))
        {
          String userSplit[] = lineValSplitOnWhiteSpace[i].split("=");
          userName = userSplit[1];
          //
          for(int j = 0; j < lineValSplitOnWhiteSpace.length; j++)
          {
            if(lineValSplitOnWhiteSpace[j].startsWith("ctime="))
            {
              String ctimeSplit[] = lineValSplitOnWhiteSpace[j].split("=");
              int ctimeEpoch = Integer.parseInt(ctimeSplit[1]);
              int differance = (int)(ctimeEpoch-startEpochTime)/interval;
              timeInterval = String.valueOf(differance);
              context.write(new Text(userName+"-"+timeInterval), one);
            }
}}}}}}
```

F.3 Reducers

F.3.1 Reduce1.java

```
package Reducers;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class Reduce1 extends Reducer<Text, IntWritable, Text, IntWritable>
{
 public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
     IOException, InterruptedException
 {
   int sum = 0;
   for (IntWritable val : values)
   {
         sum += val.get();
   }
   context.write(key, new IntWritable(sum));
}
}
```

F.3.2 Reduce2.java

```
package Reducers;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class Reduce2 extends Reducer<Text, IntWritable, Text, Text>
{
  private MultipleOutputs<Text, Text> multipleOutputs;
  @Override
  protected void setup(Context context) throws IOException, InterruptedException
  {
    multipleOutputs = new MultipleOutputs<Text, Text>(context);
  }
 public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
     IOException, InterruptedException
 {
   int sum = 0;
   for (IntWritable val : values)
   {
        sum += val.get();
   }
   String splitKey[] = key.toString().split("-");
   Text finalSum = new Text(Integer.toString(sum));
   multipleOutputs.write(splitKey[0], new Text(splitKey[1]), finalSum);
}
   protected void cleanup(Context context) throws IOException, InterruptedException
   {
    multipleOutputs.close();
   }
}
```