UNIVERSITY OF HUDDERSFIELD

An Intelligent Robust Mouldable Scheduler for HPC & Elastic Environments

Author:

Ibad KURESHI

Supervisor:

Dr. Violeta HOLMES

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Doctor of Philosophy

High Performance Computing

School Of Computing and Engineering

April 2016

Copyright

“With faith, discipline and selfless devotion to duty, there is nothing worthwhile that you cannot achieve.”

Mohammed Ali Jinnah

“Scientific thought and its creation is the common and shared heritage of mankind.”

Abdus Salam

“The desire to be rewarded for one’s creativity does not justify depriving the world in general of all or part of that creativity.”

Richard Stallman

Abstract

Traditional scheduling techniques are of a by-gone era and do not cater for the dynamism of new and emerging computing paradigms. Budget constraints now push researchers to migrate their workloads to public clouds or to buy into shared computing services as funding for large capital expenditures are few and far between. The sites still hosting large or shared computing infrastructure have to ensure that the system utilisation and efficiency is as high as possible. However, the efficiency can not come at the cost of quality of service as the availability of public clouds now means that users can move away.

This thesis presents a novel scheduling system to improve job turn-around-time. The Robust Mouldable Scheduler outlined in these pages utilises real application benchmarks to profile system performance and predict job execution times at different allocations, something no other scheduler does at present. The system is able to make an allocation decisions ensuring the jobs can fit into spaces available on the system using fewer resources without delaying the job completion time. The results demonstrate significant improvement in workload turn-around-times using real High Performance Computing (HPC) trace logs. Utilising three years of the University of Huddersfield trace logs the mouldable scheduler consistently simulated faster workload completion. Further, the results establish that by not relying on the user to suggest resource allocations for jobs the system is able to mitigate bad-put into the system leading to improved efficiency.

A thorough investigation of Research Computing Systems (RCS), workload management systems, scheduling algorithms and strategies, benchmarking and profiling toolkits, and simulators is presented to establish the state of the art. Within this thesis a method to profile applications and workloads that leverages common open-source tools on HPC systems is presented. The resultant toolkit is used to profile the University of Huddersfield workload. This workload forms the basis to evaluate the mouldable scheduler. The research includes advance computing paradigms such as utilising Artificial Intelligence methods to improve the efficiency of the scheduler, or Surge Computing, where workloads are scaled beyond institutional firewalls through elastic compute systems.

Acknowledgements

I would like to take this opportunity to thank all those who have supported and helped me along the way. I thank my parents, Mr. Nadeem Islam and Dr. Naseem Islam, who have stood by me throughout and pushed me to never settle. To my wife Anita, for her incredible support whenever I questioned everything, thank you! I would Like to thank my brother Jehanzeb, for always being there to help out.

None of this would have been possible without the support of my supervisor Dr. Violeta Holmes, without whose perseverance I may have never undertaken a research career. I would also like to thank Dr. David Cooke and Dr. Robert Alan for their guidance and direction.

I also thank all of my family and friends, of which there are too many to mention here. To name just a few, many thanks to my 'little ones', Sanam Islam and Sania Islam and my 'besties', Ahmad Khokhar and Maria Kamal.

I would like to thank all my friends and colleagues at the University Of Huddersfield, particularly the members of the High Performance Computing Research Group: David Gubb, John Brennan, Mathew Newall, Mohd El Desouki, Shuo Liang, Stephen Bonner and Yvonne James.

I would also like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

Contents

Document Word Count 43,165

List of Figures

List of Tables

Abbreviations

A.I. Artificial Intelligence

ASPP Application and System Performance Profiler

CAD Computer Aided Design

CDES Cluster Discrete Event Simulator

CentOS Community Enterprise Operating System

CFD Computational Fluid Dynamics

CPU Central Processing Unit

FLOPS Floating Point Operations Per Second

FCFS First Come First Served

GFLOPS Giga FLOPS

GPU Graphic Processing Unit

HPC High Performance Computing

HPC-RC High Performance Computing Resource Centre

HPC-RG High Performance Computing Research Group

HPL High Performance Linpack

HTC High Throughput Computing

HTCondor High Throughput Condor

JDL Job Description Language

JMS Job Management Systems

LINPACK LINear equations software PACKage

MD Molecular Dynamics

MPI Message Passing Interface

OEM Original Equipment Manufacturers

OP-EX Operational Expense

OSCAR Open Source Cluster Application Resources

PCI-E Peripheral Component Interconnect - Express

PBS Portable Batch System

QGG Queensgate Grid

QoS Quality of Service

RCI Research Computing Infrastructure

RCS Research Computing Systems

STFC Science and Technology Facilities Council

TRL Technology Readiness Level

TFLOPS Tera FLOPS

TORQUE Terascale Open-Source Resource and QUEue Manager

VO Virtual Organisation

Dedicated to future generations who can find beauty in the breeze.

Chapter 1

Introduction

With funding for research hardware declining around the world and academics favouring more flexible payment models for infrastructure, e.g. using a cloud, High Performance Computing (HPC) systems need to adapt. Shared facilities and systems utilised by diverse user groups are now a reality, and systems can no longer be optimised for certain algorithms or any single application. Previous funding models allowed research groups and projects to acquire small systems of their own. Now with a more centralised approach and shared resources, HPC systems need to cater to many different applications and algorithms.

At the same time investment in HPC and e-infrastructure is vital as it leads to economic growth and helps accelerate research and development. A UK government report from 2012 has stated:

> *"High Performance Computing (HPC) and e-infrastructure in general are drivers of economic growth and societal well-being. They are also vital for maintaining international competitiveness in the generation of knowledge and its application."*

(Kenway, 2012)

Energy efficiency and a reduction of carbon footprints is now a cornerstone of many IT related business and sustainability plans. System administrators and IT managers are constantly trying to maintain a balance between system efficiency and quality of service. To meet the various requirements of the user community, the system scheduling parameters have to be loosely configured to allow flexibility. However to get the maximum efficiency out of a system it must be governed with stringent rules that would result in maximum utilisation. Users only see the quality of service and find a system useful when just the right balance is struck. This balance depends on the applications behaviour and characteristics.

Reiterating the importance of Research Computing Infrastructure (RCI) in academia and industry, the UK Minister of State for Universities and Science stated:

"UK universities are tremendous drivers of the economy and our success is crucial to the nation as a whole ... But we know we need to use our resources in better ways to deliver maximum efficiency. One key innovation detailed in the report is the emergence of asset-sharing arrangements within groups of universities, such as the N8 and M5. These schemes allow institutions access to research equipment across the groups, maximising their usage and also opening up university facilities to industry"

(Kelly, 2014)

1.1 Background

Between October 2009 and August 2010 the University of Huddersfield launched an ambitious project to setup a vast RCI, which would serve the University's research community by providing robust computing solutions. These solutions

took into account the various kinds of requirements different fields have, with regards to computing, data processing, visualisation, and the tools available to meet these specific needs. During the 10 month project the High Performance Computing Research Group (HPC-RG) was both formed and set about positioning the University of Huddersfield as a recognised institution in the field of parallel and distributed computing.

Several HPC clusters were deployed along with multiple High Throughput Computing (HTC) resources. These various resources were then unified to form a monolithic campus grid known as the Queensgate Grid (QGG). This computational grid is heterogeneous and caters for Art and Design students and researchers who need Windows or *NIX platforms for 3D rendering; to mathematicians and physicists who need to processes many thousand's of small calculations; and then engineers and chemists who need many processors to carry out simulations that even the most advanced workstations are unable to handle. These systems have fed into the teaching and learning environment as well, so they are not just seen as purely research equipment but also make up a part of the undergraduate and postgraduate student experience. (Bonner et al., 2013)[1].

Being a small-medium University with a cap on space and power, the local RCI can not be scaled to the levels seen in large HPC centres. To meet the ever growing user requirements the University of Huddersfield grid was then interfaced with the then UK National Grid Service and the North East Grid. This gave researchers access to national supercomputers and led to an acceleration of research within the University. Huddersfield was also able to feedback into these National Grids as the QGG brought many softwares that were previously unavailable on the network.

[1] A second paper about introducing HPC into a teaching environment is ready for submission to an appropriate call.

Feedback on publications (Kureshi, 2010; Holmes & Kureshi, 2010) relating to the establishment of this Grid included:

“This is an interesting case study of rolling out a grid in a campus environment; it will no doubt be of interest to both researchers and IT professionals on many such campuses...”

By the end of 2010 a platform was established for the HPC-RG to begin research in new computing paradigms like co-processors, elastic computing and data centric computing. To this end a GPU cluster, a private Cloud, and an Apache ®Hadoop cluster was deployed to tackle research challenges (Bonner et al., 2013; Newall, Holmes, & Lunn, 2014; Kureshi et al., 2013; Bonner et al., 2014).

To meet the internal demand for more computing power, the University of Huddersfield bought into a shared HPC facility at the Science and Technology Facilities Council (STFC) Daresbury. Keeping in mind the power and space limitations, and the need to reduce the carbon footprint, switching to this Operational Expense (OP-EX) model was the most feasible solution. While modern cloud computing is usually associated with the term pay-as-you-go, the arrangements in this deal are more OP-EX as the arrangement was to pay for a fixed number of CPU hours (CPU-hrs) on a monthly contract, with a percentage rollover of unused hours from month to month.

During the lifetime of this agreement the author and other system administrators at Huddersfield began to notice high-levels of *”bad-put”* (bad-throughput) in user workloads. At a most basic level bad-put from the users would lead to over subscribing (or under booking) resources and causing system down time. This system was shared between several customers (commercial and academic) and such downtimes are frowned upon. At a price-performance level user over bookings cost the University hundreds if not thousands of Pounds. If a user

booked 12 cores for 12 hours but only used 1 core, the University was still billed 144 CPU-hrs.

One of the exciting features of being part of the shared facility is that when certain sections of the system were unused the University of Huddersfield could extend its load to the entire system. However bad-put of expected program run times meant that scheduling, for these extended usages, was near impossible. One method of countering the bad-put that was employed was to limit the overall time a single job could run. Therefore if Huddersfield users were occupying the whole system and another customer submitted a job, a guaranteed start time could be presented to the customer. But this adversely affected users whose simulations required long run times and could not be check-pointed to restart at a later time.

Similarly on the internal HPC systems large jobs would either jam the queues waiting for resources and leave the systems idle for long periods, or if out-of-order execution (explained in Section 4.2) was enabled, the large jobs would get stuck behind smaller jobs leaving users unhappy with the over all Quality of Service (QoS). On closer observation the bad-put observed on the shared facility existed within the QGG as well, but because the system and job sizes were smaller and as there was no direct cost associated to using the system, these bad-puts went largely unnoticed.

In order to negate the bad-put it was felt that automating the process of resource allocation was required. To do this the system would need to know several key things about the job at submission time: the performance characteristics of the application (on the system), and the size of the users workloads. Using this key information a scheduler could be devised to make resource allocation decisions and execute out-of-order job processing without adversely affecting the quality of service (QoS) delivered to the end user. The term quality of service within this

thesis is derived from the turn around time for a job. Upon submission of a job its time to completion can be calculated by adding the execution time and the time spent waiting for the required resources to become available. If the job sticks to this schedule despite out-of-order execution, where a job submitted later runs first, then the QoS is considered to be good/unaffected. However if jobs end up waiting longer in queues due to resources being allocated to out-of-order jobs then the assumption here is that the quality of service has decreased. The work to develop this novel scheduler was carried out between January 2011 and December 2013 at the University of Huddersfield.

1.2 Aim

This thesis aims to explain novel approaches to automating resource allocation in order to facilitate better utilisation of HPC resources, with minimal operator input and maintain the QoS for the user.

Using the University of Huddersfield research computing environment (its infrastructure and its drivers) this thesis describes a mouldable scheduler that utilises application performance profiles to efficiently allocate resources and predict end times for running jobs. This ability to look into the future will allow for better *re-ordering* of queues to maximise both the efficiency and the observed QoS.

1.3 Objectives

The objectives of the project are to:

1. Design and develop a scheduling system to take into account application performance;

2. Devise a mechanism for a system to determine the performance characteristics of an application. The benchmarking system should take into account the size of the input data and then feedback expected run times versus required resources to the scheduler;

3. Evaluate and develop methods to simulate HPC environments to test the new scheduler. The simulator should be able to analyse realistic workloads to assess the true effectiveness of the system;

4. Investigate existing workload management systems, in industry and in the theoretical realm to incorporate best practices during the development of the scheduling system;

5. Investigate research computing environments to determine where the shortfalls noted in Section 1.1 can be observed. This investigation will include changes in scheduling requirements in emerging computing paradigms (e.g. clouds);

6. Analyse scheduling techniques and practices to understand the best algorithms which aim to meet the needs of utilisation vs. QoS to integrate within the mouldable scheduler.

1.4 Outline of Work

This thesis is structured to match the chronological order of the mouldable scheduler project's research and development activities. In the first half of the project an exhaustive analysis was undertaken covering scheduling techniques, scheduling systems, distributed computing paradigms, and advanced computer management techniques. As the problem of bad put was observed within the University of Huddersfield's own workload, an in-depth analysis was carried out

on the nature of applications and the characteristics of all jobs that were executed on the local RCI.

In order to achieve the aim described in Section 1.2, development on mechanisms for analysing application and system performance was carried out after the background study. In parallel a simulator was also developed so that a stable testing environment was available for evaluation. With the simulator and benchmarking tools ready, data was collected on application performance and system utilisation. The mouldable scheduling algorithm was then developed and implemented within the simulation environment.

All code, reports, and documentation in this project has been version controlled and archived in the central repositories of University of Huddersfield HPC-RG[2].

Chapter 2 outlines research computing paradigms and aims to create a broad picture of how these complex computer systems work. The evolutionary path these systems take, shows the challenges and requirements of industry and end-users. This is followed by a chapter explaining how Job Management Systems (JMS) work and what is required of them. It elaborates on the mechanisms whereby divergent user requirements are balanced against limited resources available within Research Computing Systems (RCS).

A detailed review of publications relating to benchmarking, scheduling, and intelligent systems is presented in Chapter 4. Furthermore different approaches to scheduling in traditional rigid environments and previous attempts at mouldable scheduling are also explained in detail. Additionally a close look is taken at scheduling in new and emerging computing paradigms.

Chapter 5 builds on Section 1.1 and explains the University of Huddersfield's RCI. A detailed outline of usage characteristics of one of the HPC systems is

[2]http://repo.qgg.hud.ac.uk/

also presented. The actual user workloads presented in this chapter form the input used to test the various components designed in this project.

The Application and System Performance Profiler (ASPP) explained in Chapter 6 is the result of designing and deploying an open-framework benchmarking suite for HPC systems. This suite allows a system administrator to plug-in a real application, with associated datasets and workloads, to benchmark the systems. The suite autonomously creates a parameterised sweep of the datasets over the system, to determine the application's performance using different HPC resource allocations from the system. Deployed against the Terascale Open-Source Resource and QUEue Manager (TORQUE) JMS, ASPP can interpret job logs from each parameterised run and generate a knowledge base. These results are stored in a database for further use but can also be outputted as performance characteristic curves using open source statistical packages.

Chapter 7 presents the Cluster Discrete Event Simulator (CDES) as a strong candidate for HPC workload simulation. Built around an open framework, CDES can take system definitions, multi-platform real usage logs and can be interfaced with any scheduling algorithm. CDES has been tested against 3 years of usage logs from a production level HPC system and verified to greater than 95% accuracy.

The design and testing of the mouldable scheduler (shown in Figure 1.1, revisited in Chapter 8) is described in detail in Chapter 8 . The scheduling algorithm is implemented within the CDES simulator and utilises the application profiles generated by the ASPP. The scheduling decisions outlined in this chapter are rigid or rule based and job resource allocations are evaluated at execution time. The mouldable scheduler is driven at a base level by a First Come First Served (FCFS) algorithm. The conclusions of this chapter include current observed limitations of the mouldable scheduler.

Chapter 9 outlines further work carried out using the mouldable scheduler's components and is divided in two parts. The first part (Section 9.2) explains how fuzzy logic can be used to tackle the limitations observed in Chapter 8. The intelligence introduced into the system by the fuzzy logic opens the door to out-of-order scheduling of jobs. The second part of this chapter includes developments in surge computing. Surge computing allows for traditional HPC systems to scale up into elastic compute services, driven by heavy loads or to meet specific requirements by users.

Chapters 11 and 10 cover further work being carried out and the conclusions of this thesis, respectively. The motivations, direction and implementation of a strategy for a mouldable scheduling system will become apparent at the conclusion of these chapters.

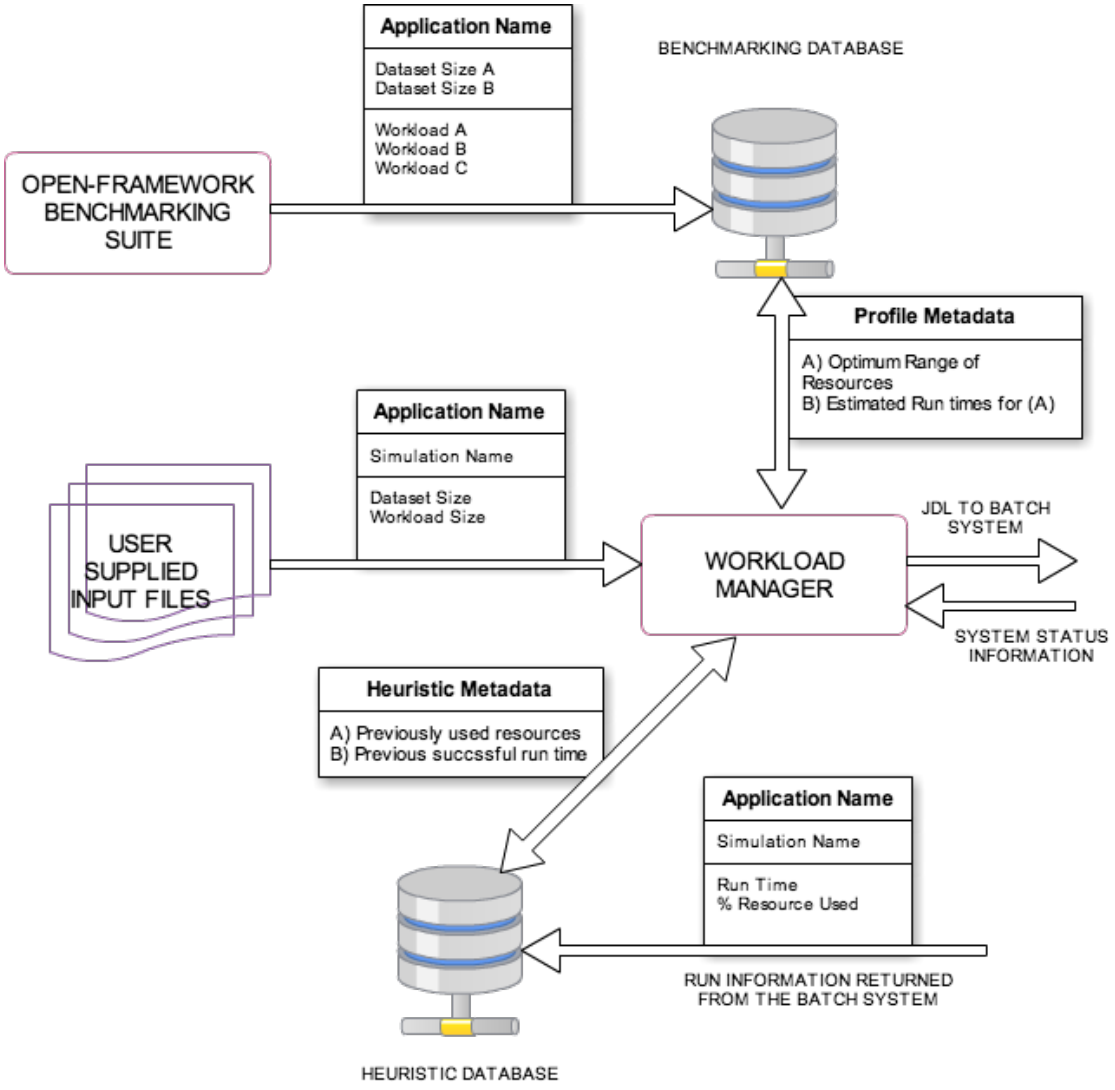

FIGURE 1.1: The proposed Mouldable Scheduler

Chapter 2

Research Computing Systems

2.1 Introduction

The term Research Computing Infrastruction (RCI) refers to Information Technology (IT) infrastructure that is used to do scientific research. While this can include the individual laptops, desktops and workstations available to a researcher, the term Research Computing System (RCS) generally implies ”big metal” systems. Racks of powerful servers or buildings full of workstations all coming together to work on one problem make up the main thrust of any institution’s RCI.

As science has progressed the computational requirements have grown exponentially. Scientists aim to reproduce the living-world as computational models to test new hypothesis. These models are used in many disciplines such as weather forecasting, earthquake predictions or urban planning. To get these models as accurate as possible a researcher requires more storage, memory and processing power than is available in workstation or desktop configurations.

The analysis of automobile aerodynamics using Computational Fluid Dynamics (CFD) is a good example of how a problem increases its computational demands as researchers make their models more realistic. The more accurate the simulation required, the more compute power is required. Accurate here means more degrees of precisions and an analysis over a finer sampling time. Simulations with high degrees of precision would take days to execute on an ordinary workstations. Using RCS this computational time can be reduced to a matter of hours. If a researcher wants to precisely model their automobile, the granularity of the mesh or surfaces of the model grow. Larger mesh sizes need more working memory to process the data. Workstations and desktops struggle to meet the RAM requirements of a fine grain to-scale model of an automobile. Finally, any design of an automobile or its components, needs to be tested and simulated under different conditions. As the granularity of the model and the accuracy of simulations increase so does the size of the data files. The larger the number of simulations performed under different conditions the more subsequent data is generated. Eventually this data will not fit on the standard desktops and conventional storage, slowing down file access and increasing search times.

To satisfy this need for higher processing power, larger memory and greater storage, High Performance Computing (HPC), High Throughput Computing (HTC) and other Research Computing Systems (RCS)'s are required. In Sections 2.2 and 2.3 traditional High Performance and High Throughput configurations of research infrastructure are discussed. This is followed by an analysis of a new computing paradigm - *cloud computing*, that is now being adopted within RCI's, in Section 2.5.

2.2 High Performance Computing (HPC) Systems

High Performance Computing (HPC) systems refer broadly to supercomputers or computer clusters. These are large computer systems made up of many compute cores and large amounts of RAM. Cluster Computers are a collection of servers tightly integrated using fast network interconnects. Supercomputer refers to proprietary systems with custom designs and interconnects, typically with large amounts of memory and Central Processing Unit (CPU) on an integrated circuit. They are not modular like compute clusters. As of June 2014, Compute clusters make up over 80% of the world's top 500 supercomputers (J. Dongarra, 2014).

Small computer clusters can be built simply by combining commodity of-the-shelf workstations networked by an ethernet connection. At the higher end clusters can be a set of highly optimised and dense servers linked with high bandwidth, low latency fibre optic interconnects. Systems based on commodity of-the-shelf hardware are known as Beowulf clusters. Commonly a Linux family operating system is deployed on the system. Linux clusters make up more than 90% of the worlds top 500 clusters (J. Dongarra, 2014).

IBM's Bluegene, Cray's XK7 and the SGI UV are examples of modern super computers. The current iteration of the IBM super computer is known as the Blue Gene/Q. This system uses IBM's proprietary processor architecture known as Power PC. Special interconnects link each processor electrically in a 5D Torus configuration to create a 'node' of 512 processing elements. The Cray XK7 uses conventional x86 processors from AMD but use their own proprietary interconnect to link the cores. This system is called Gemini.

The SGI UV forms a different class of supercomputer known as Shared Memory

systems. Due to proprietary design these systems are still considered supercomputers but they differ from the previous two examples in that they generate a single system image where large banks of memory are connected. Shared memory supercomputers create a large addressable memory space for a single operating system to reference directly.

Cluster computers are mostly distributed memory systems composed of general purpose servers. Broken down to their base systems each node can be used for general IT infrastructure purposes. Due to advancements in x86 architecture processors, network interconnect technologies, operating system and middlewares, computer clusters can be as fast, if not faster that supercomputers (Sloan, 2004).

A middleware is used to harness the power of the individual systems or nodes and give the user the illusion of one very big system. A middleware is a suite of software that are used to tightly couple the disparate resources to create a single system image. Typically a middleware has some mechanism to put a working operating system on every node; synchronise user information and data across all nodes; enforce security and policies and finally to organise and deal with multiple users. The latter is the purview of a Job Management System (Sloan, 2004).

Using a special set of libraries defined by standard Message Passing Interface (MPI), software can be written to harness the power of the many cores in a cluster. MPI routines allows remote processors to communicate directly with each other. Remote here implies that those cores do not share space on a silicon chip or motherboard, though MPI can also be used for on-chip communication (Sloan, 2004).

HPC systems are not always used to process a single problem. Due to licensing or the limitation of software a single simulation may not be able to use all the

processors on a large HPC system. Consequently the job management system segregates cores for different jobs and restricts the MPI 'world'(Sloan, 2004).

Supercomputers and clusters can be designed for specialist applications. The SGI Altix supercomputer is designed for graphic and large image rendering. These systems are connected to large video arrays and the end user gets to directly interact with the system. Another major field that is rekindling the HPC world is big-data analysis. Software like Apache Hadoop (developed by Yahoo) utilise fast storage, memory and processor arrangements to process large quantities of data. Some computer clusters are designed specifically to handle only Hadoop and therefore use specialist middleware, but traditional clusters can also be configured (with direct intervention of the job scheduler) to do big data analysis (Sloan, 2004).

2.3 High Throughput Computing (HTC) Systems

A second paradigm within research computing is that of High Throughput Computing (HTC). Small distributed systems are linked to handle simulations that are "embarrassingly parallel". The problems that can be broken down into many discrete parts are ideal for high throughput computing environments. It is the divide and conquer approach that allows large problems to be quickly solved, thus lending to the HTC name. HTC differs from HPC, in that HTC handles discretised problems and they systems are loosely coupled.

In an academic or corporate setting, the HTC paradigm offers a low cost of entry into research computing. These two environments tend to have many workstations across their real estate. When students/faculty or employees are not at their desks these machines are idle and yet consuming power. Harnessing

these idle systems is the easiest way for any organisation to build its Research Computing Infrastructure (RCI).

HTC middleware is required to manage very dynamic environments. An idle node may not remain in that state for long and unlike an HPC system, the load can be introduced external to the middleware. Also, there could be a poor network connection between the primary system and the remote machine. Finally one of the biggest differences of an HTC system, compared to an HPC system, is that the working environment can be totally heterogeneous. It is not uncommon to have Windows, LINUX and OSX systems in a single pool of systems. The HTC middleware has to ensure that appropriate executables and data are migrated to the end point.

Two popular middleware for HTC based research computing are HTCondor and BOINC. The Berkley Open Infrastructure for Network Computing (BOINC) is a high throughput middleware that is commonly used to crowdsource compute power for research projects (Anderson, 2004). It is a small tool that the public can install and set the rules as to when the middleware can operate. Once configured, it is up to the downloader to choose to which project they would like to contribute resources. When the use conditions are met, the public computers connect to the head node of the project. The head node can transfer executables and data to the remote nodes. BOINC is used by projects like Folding@home (related to DNA folding) and SETI@home (Search for Extra Terrestrials)

HTCondor is developed by the University of Wisconsin and is a very mature piece of software. HTCondor is ideal for trusted single ownership environments. HTCondor is able to group multiple machines together to support parallel processing similar to HPC environments. Multiple Condor pools that are geographically disparate can be linked together forming a grid.

2.4 Grid Computing Systems

Grid computing brings elements from HPC and HTC technologies together to integrate traditional research computing provisions. Computational grids are composed of geographically remote HPC and HTC systems that are linked together through the internet. Usually owned by different entities, access to the disparate systems is enforced by service level agreements (SLA), leading to the formation of a Virtual Organisation (VO). Gird computing can be used to harness the power of multiple HPC systems to tackle one problem or can be used for high throughput distribution of high performance type jobs across multiple systems.

Differing from HTC systems, the grid middleware needs to be provided with information (usually a Uniform Resource Locator (URL) or Internet Protocol (IP) Address) about all the execution end points. The grid middleware on the local system connects to the installation of the grid middleware at the remote site to get system information. These grid middleware sit on top of the HPC or HTC middleware. Therefore between an end user and an execution endpoint there are now two distinct middleware.

The Globus grid middleware is a popular middleware used to link HPC and HTC systems around the world. Globus was developed in the United States at the University of Chicago (Foster, 2011; Allen et al., 2012). The Globus Toolkit includes packages to handle resource reporting, file transfer, remote terminal access and remote job submission. The gLite middleware is developed in Europe and is the middleware of choice for the Conseil Europeen pour la Recherche Nucleaire (CERN) project (Laure et al., 2006). gLite is able to directly interact with HPC and HTC schedulers (like PBS and HTCondor) and it is able to interface with other grid middleware including Globus. gLite also has the advantage of having a modicum of resource discovery.

Grids were envisioned as the answer to provide end-users computing power as easily as plug and play. The name and inspiration comes from electric grids, where an end user can plug in their appliance and the power is instantly provided. In case of electric power the user does not need to know if the power comes from a nuclear or solar power source. Similarly on a computational grid the end-user would use compute power, regardless of its source or location.

Unfortunately, due to software limitations and licensing along with the overheads of SLA's, grid computing was not able to deliver that "fourth utility" aspect the developers and scientists had hoped for. This title was taken by a new computing paradigm 'Cloud Computing'. Building on the power of HPC and HTC systems and the resiliency and distributed nature of the grid, Cloud Computing delivers IT infrastructure to the end user over the internet, using a commercial (pay for a service) model.

2.5 Elastic and Shared Systems

Centred on Service Oriented Architecture (SOA) Cloud computing has many facets. As everyday operations move towards being "internet ready" it is very difficult to limit those services that actually form the cloud. It appears that there is no fixed definition for a cloud and any on-demand internet service becomes a cloud service. This lack of definition reflects in the name. A definition provided by the Ian Foster et al., the authors that wrote the book on Grid computing (the predecessor to clouds providing on demand infrastructure), states:

"A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted virtualised, dynamically-scalable, managed computing power, storage, platforms and services are delivered on demand to external customers over the internet."

(Foster, Zhao, Raicu, & Lu, 2008)

This definition while giving a good generalised overview can now be attributed to a public cloud infrastructure. Currently many services being offered under the banner of cloud computing may not be virtualised and are surely not dynamically scalable. A paradigm that this definition does not acknowledge is the private cloud. Private clouds are devoted resources belonging to a single organisation that provide on-demand resources to its internal user base. They are usually used as an easy method to increase capacity or provide testing environments with minimal effort (Grossman, 2009).

While there is a dispute on the definition of a cloud, there is consensus that there are three major services – IaaS, PaaS, and SaaS. In an ACM article IaaS is defined as Infrastructure-as-a-Service, PaaS is Platform-as-a-Service and SaaS is defined as Software-as-a-Service (Armbrust et al., 2010). Software-as-a-Service refers to the delivery of a software package over the interface via a web browser interface. The actualy compute power is on the server in a remote datacentre. Good examples of SaaS are Google Apps™, Force.com and Facebook. Platform as a Service closely resembles an Operating System over the Internet where users or developers can experiment on a fixed configuration of infrastructure. This is very popular with application developers and commercial providers of PaaS include Google® App Engine™ and Microsoft® Azure™. IaaS is facet of cloud computing that can be defined as the ”killer application” for cloud computing. The ability for a company, with just one computer in their inventory to be able to set up a vast datacentre, meeting all their business needs without needing the real estate, tenders, power etc. is where the hype lies. Commercial providers of IaaS include Amazon's® EC2™ and S3™, Ubuntu One™ (Weiss, 2007; Foster et al., 2008; Armbrust et al., 2010; Vaquero, Rodero-Merino, Caceres, & Lindner, 2008; Dikaiakos, Katsaros, Mehra,

Pallis, & Vakali, 2009; Hayes, 2008; L. Wang et al., 2008; A Vouk, 2008; Buyya, Yeo, Venugopal, Broberg, & Brandic, 2009).

The Foster el al. definition also leaves the term ”delivered on demand” very vague. There is a school of thought that believes that if there is human interaction between the demand and the delivery of the service then this is not cloud computing and should be instead classified as an e-commerce activity (Weiss, 2007). The terms economies of scale and dynamically scalable from the Foster et al. definition seems to tend to the same concept of the system being fully automated. For the definition of Infrastructure as a Service cloud we will adopt the Foster definition with the caveat that the service must be virtualised, full automated and can include private resources or commercially available public resources. This was enforces by David Wallom in his presentation about the NGS cloud pilot (Wallom, 2010).

Cloud computing has been described as the next paradigm in computing. It promises to deliver computing power as the fourth utility direct to the end user. While large enterprises have yet to move their IT infrastructure into the cloud, startups and small businesses have benefited greatly from the cloud. The shorter barrier to entry means that researchers can get access to computational power just by swiping their credit card (Foster et al., 2008).[1]

Small to medium enterprises (SME) have successfully employed cloud computing in scaling their existing infrastructure, as and when required. Most organisations maintain their internal infrastructure but during periods of heavy load the ICT footprint can be scaled beyond the institutional firewalls. The companies can therefore guarantee a high quality of service while limiting the amount of capital investment. This form of provisioning is known as surge computing (Van den Bossche, Vanmechelen, & Broeckhove, 2010).

[1] This work has been published in the International Journal of Advanced Computer Science and Applications (Kureshi et al., 2013).

In their paper Scientific Cloud Computing: Early Definition and Experience, Wang et al. outline how cloud computing can be used for scientific research and how this relates to a QoS guarantee to the end-users. This effort moves away from the traditional use of clouds from an implementation of infrastructure for general IT purposes to an implementation of e-Infrastructure for academic and scientific purposes (L. Wang et al., 2008). It should be noted here that Wang et al. incorporate system up-time, access to faster processors and scalability in their definition of Quality of Service (QoS) i.e. it is not only the wait time of the job but also the run time of the job that they hope cloud computing will improve. To the end user of HPC in the cloud Brandt et al. use the notion of a private cloud to deploy what can only be described as an HPC system in a cloud sitting on an HPC system. Their findings conclude that the just-in-time provisioning of hardware and software helps create a more flexible system to meet the needs of a diverse end user group (Brandt et al., 2009).

From there the concept of *surge computing* has also emerged. Surge computing is used when a site's HPC resources have reached capacity consumption and needs to scale up to a private cloud or a shared resource and then out to a public/commercial cloud (Marshall, Keahey, & Freeman, 2010). While these concepts have been executed in the past the approach has been rigid and it takes a system administrator to realise that a system is at capacity, provision the resources in a cloud and make the connection to the local site. If the Wallom and Foster definitions are to be followed to the letter this would not classify as cloud computing as there is human interaction between the end-user executing his/her job and the resource becoming available.

While the industry is primarily focussed on surging Web 2.0 workloads (Chieu, Mohindra, Karve, & Segal, 2009; Vaquero, Rodero-Merino, & Buyya, 2011; Mao, Li, & Humphrey, 2010) there are many who advocate migrating HPC

workloads to the cloud (Nurmi et al., 2009; He, Zhou, Kobler, Duffy, & McGlynn, 2010; Bientinesi, Iakymchuk, & Napper, 2010). However, there has been little existing work in creating a HPC Job Manager and Scheduler that is truly dynamic and elastic. The HTCondor project is a dynamic HPC scheduler, but it lacks elasticity, working on the principle that execution nodes can start up and then connect to the control node. The control node itself does not 'hunt' for execution endpoints or tries waking end points. However the dynamism in HTCondor can be utilised statically for HPC type workloads in the cloud. It has been utilised by pharmaceutical companies to create large clusters on Amazon EC2 (Brodkin, 2011). The major limit to the dynamism currently provided is that based on the workload on the head node there is no mechanism to automatically generate execution endpoints.

There are several commercial ventures that use existing IaaS systems as a platform to offer High Performance Computing as a Service (Brodkin, 2011; Trader, 2012). These companies offer virtual instances of popular HTC tools that will work in the cloud environment. Developers of job management systems have also paid close attention to this "fifth utility" (Buyya, Yeo, & Venugopal, 2008) and are in the process of developing next generation tools that will scale automatically (Wilson, 2010).

Traditional HPC Job managers are very rigid. Job managers like TORQUE and Grid Engine need to be given endpoint information, e.g. hostnames or IP addresses, at startup. If a change is required then the whole job management suite needs to be restarted. This obviously makes such schedulers inelastic. Recently however, IBM (in collaboration with Platform Computing) have released a version of the Platform LSF that dynamically creates nodes within elastic environments to meet such HPC needs. The Enterprise Edition of the MOAB HPC Suite now also includes dynamic cloud based provisioning functionality. Penguin Computing too have a cloud based initiative for enterprise

clients (Bernstein & McMahon, 2012).

2.6 Security

Each of the systems outlined in this Chapter have different approaches to security. For HPC systems the head node of the cluster is the primary gatekeeper and all security exists on it. Within the cluster there are typically no firewalls or restrictions between nodes and with the head node. This is because when a job runs on a subsection of the cluster one of the nodes assumes the role of master and needs to execute instructions on other remote nodes. As a standard practice internode communication takes place over Secure Shell and so the instructions are encrypted preventing users snooping on each other. However as HPC systems typically tend to be own and utilised solely by organisations there is an implicit trust between the users. Further layers of security will add to the latency of the system. The head node typically has firewalls, encryption and strict user access lists to ensure that no external sources can breach the systems security. The internal network is kept strictly segregated.

HTC systems take security considerably more seriously as the internode communication and data utilises public networks (although these still tend to be within the organisation). Encryption between internode communication is common and nodes have access control lists to ensure only authorised nodes are connecting in. As bag-of-task type workloads are run on HTC systems there is little internode communication and nodes only need to be concerned with the master node, the scheduler node and a small list of submit nodes (these services could all be encapsulated within one node).

Grid system use the internet to communicate and access by non-organisational users is a requirement there is much stronger encryption and two-way identification. Users and data are secured using X509 certificates and encryption enhancements to openSSH (Novotny, Tuecke, & Welch, 2001). A trusted 3rd party certificate authority generates host certificates to identify all systems and access points within the grid. User certificates are also generated so that users can be uniquely identified to their organisation. This way the authentication of users from different organisations can be done and depending on the SLA's agreed between each site the authorisations can also be managed.

Cloud computing has adopted a reduce level of security as compared to grids. The 3rd party driven two-way authentication and SLA based service was deemed to restrictive for wide scale adoption. Beyond payment and account management webpages (that are encrypted based on typical SSL encryption) security on the cloud is what the end users wants it to be. Security of point-to-point communication must be handled by the end user. Commercial contracts and the threat of legal action protect users data while it is in the service providers data centre. For users deploying surge environments in public clouds encryption of their data from the local system to the remote instance will need to be the primary concern. This can be handled through the encryption and authentication tools with network files systems.

The scheduler being described in this thesis sits within the application layer (right on top) of the 7 layer Open Systems Interconnection model (OSI model) and manages other applications. It relies on external systems for security that are usually provided by the operating system. Within HPC systems, whether deployed on physical infrastructure or in the cloud or across both, scheduling decisions are not made on the head node itself. Jobs are submitted to the queue and the scheduler decides the order of the jobs within the queue. The job management system then delegates the job. The scheduler does not need

to take into account security as these are handled by the Job management system and the operating system services.

2.7 Summary

In this Chapter an outline of the different parallel, distributed and elastic computing paradigms have been presented. A key element to these RCSs is that they are all typically shared computing systems, i.e. enabling more than one user at a time, and usually consume large amounts of power. What is also interesting to note here is that computing paradigms appear to be cycling back to large compute resources centrally located with low powered satellite devices as it was in the period preceding the microprocessor.

Due to the shared nature of the system users workloads have to be encapsulated as discrete jobs. Due to the power requirements and general QoS for the end users these jobs have to be scheduled appropriately to ensure maximum QoS and minimum power consumption. In the next two chapters the characteristics of job scheduling systems (3) and techniques (4) will be explained.

Chapter 3

Requirements of a Job Scheduler

3.1 Introduction

Since the early days of computing, Job Management Systems (JMS) have been essential to the operation of large compute systems. Initially job management systems were called batch processing systems. Modern computing differentiates between batch processing system and job management systems, where the batch system is part of the job management system. In this chapter the importance of these systems and their evolution is outlined. Section 3.2 covers batch queuing systems and job schedulers. In Section 3.3 the layers of complexities associated to scheduling in different computing paradigms is outlined. In Section 3.4 the features and functionality of publicly available schedulers is reviewed. In the last Section 3.5 a brief description of exisiting workload simulators is presented.

3.2 Job Management Systems

Job management systems are divided into two primary components; Batch Queuing systems that accept execution jobs from users and assign them to resources on the underlying system when available; and Job Schedulers which determine the order of execution for each job.

3.2.1 Batch Queuing Systems

Batch Processing systems can trace their history back to early mainframes (Tanenbaum & Tannenbaum, 1992). Data used to be input into systems using punch cards and similar ”slow” mechanisms. The mainframes were not utilised in an online manner as it was not cost effective. For several hours operators would upload data from tapes and punchcards into the system and then the computer would process all the inputed data in a batch. Each discrete simulation or program within the batch is commonly known as a *job*. With the advent of the modern microcomputer, users have become used to a real time response from their computers. As mainframe computers have evolved into super computers and clusters, they have migrated from offline processing to online processing.

Under this new approach to running large systems, batch processors are used to effectively share the resources between multiple users. As modern High Performance Computing (HPC) systems support multiuser environments the batch processing systems' primary function is to ensure that system resources are not over subscribed. Thus batch processors can now be considered queuing systems. Jobs are processed as they arrive but when the system becomes over subscribed jobs begin to be queued. This class of software is still known as a *batch* system because it is responsible for the execution of many jobs at the

same time. As mentioned in Section 2.2 modern systems tend to be very large and not all applications can scale-up on these resources. The batch queuing systems are responsible for dividing the system and assigning the appropriate parts to different jobs. Commonly a user may submit more than one job at a time. Since each job only needs a subset of the HPC system the batch processing system becomes a good tool for users to manage their workloads.

Modern batch systems are defined as online systems because the moment a job is submitted to the system it is considered to be executed. As discussed previously when describing the offline batch system, all jobs in the batch had to be uploaded before execution would begin. However, modern batch processors are not considered real-time systems because under most conditions after the user submits the job, they are unable to determine the state of the program or its output until the execution cycle completes.

3.2.2 Job Schedulers

Once a job has been submitted to a batch queue the order of execution is determined by the job scheduler. While first-come-first-served maybe the most intuitive method to order the execution, it is not always the most effective solution. There are several reasons why system administrators may want or need out of order execution. While scheduling algorithms are discussed in detail in Section 4.2, this section covers the reasons for out-of-order execution and explains why the scheduler is an integral part of the job management system.

Large compute or HPC systems consume a lot of power and require special cooling systems. Even when idle, large clusters tend to consume electricity in megawatts (J. Dongarra, 2013). System administrators aim to maximise the usage of these systems, so that while powered up the machines are always fulfilling their purpose rather than sitting idle. There could occur a scenario

where an HPC system is under 80% load and the next job requires 30% of the system. Under a first come first served approach 20% of the system will sit idle awaiting the 10% required to begin the next job. However, in First Come First Served (FCFS) with back-filling, if the second or third job in the queue require less than 20% then these jobs can be executed first. This way the system is always well utilised and for all the electrical power delivered in, the system delivers computational power out.

Backfilling leads to issues of Quality of Service (QoS). Very large jobs will get deferred in favour of small jobs and this will slow down users' work. Schedulers are therefore required to ensure that a certain QoS is maintained. This QoS can be extended to include higher priorities for users or projects. Users may have deadlines, or in case of commercial settings users could be paying for their time on the system. The jobs submitted by these users can not be deferred for business reasons. Job schedulers are required to handle such decisions.

As mentioned in the previous section, a single user can rely on a job scheduler to mange his/her workflow. Some applications may have multiple stages within their execution cycle. A user could submit two jobs where the second job depends on the data generated from the first job. Without a scheduler the batch queuing system will execute both jobs as soon as there is a resource available. The scheduler however, will be able to hold the second job, awaiting the result of the first. While holding the second job, other jobs that arrive on the system can be executed out of order.

Chapter 4.2 details the different algorithms that have been developed to better manage large computer systems, leading to better QoS and resource utilisation.

3.3 Scheduling Decisions

In Chapter 2 various different paradigms of research computing were outlined. Each evolving paradigm (HPC to High Throughput Computing (HTC) to grids and clouds), brings about its own challenges that job schedulers are expected to manage. The more components there are to be managed within a system, the more complex the scheduling problem becomes. If an element of dynamism is added, the complexity increases even further.

Even in a single user system there is a small element of scheduling. All modern personal computers support multitasking and the operating system must manage which compute cycle is allocated to which process. If done incorrectly the user experience is severely hampered.

HPC systems, such as computer clusters, must also deal with multitasking at node level. The multitasking on a node has to deal with running the application on a node level as well as the cluster level, responding to the Job Management Systems (JMS) polling for node health and job status. This node level scheduling is then augmented by the cluster level scheduler. Since a cluster appears as a single system, the scheduler within the JMS provides the multitasking element at a macro level.

Within HTC environments the job schedulers have a different but still complex problem to deal with. As HTC systems tend to operate on the principal of scavenging, a scheduler has to deal with the added complexity of an execution end-point not being available or totally disappearing. The heterogeneity of the HTC environment also poses additional challenges. An HTC scheduler needs to determine which node or execution end-point is the most suitable for the current job. Some of the questions that need to be considered are:

- Does a node meet all the requirements for the current job?

- Is the node reliable with its up time?
- Can the required data be transported to the node?

These types of constraints have to be weighed against the perceived **QoS!** (**QoS!**) that needs to be delivered to the end user.

Grid Computing further compounds the complexity of the problem. Grid schedulers are expected not only to be concerned with the job occupancy of tightly coupled systems, but also with dynamically variable environments similar to those present in HTC environments. Grid schedulers do not have complete control over how jobs are executed at each remote site. Primarily, the grid scheduler has to overcome the fact that the users may submit jobs directly at/to the end-point, and therefore the load at each end-point can dynamically change at a moments notice. Jobs submitted to the grids may be executed on multiple systems simultaneously. Each end-point can be geographically very remote or can have different local scheduling policies. The scheduler needs to factor the in-flight time. In-flight time refers to the time taken for the job and associated data to reach the computing end-point. To further complicate the matter a single job may need to use compute cores over two sites simultaneously. The scheduler in a grid needs to know the eccentricities of each end-point and must have sufficient interoperability to talk to multiple remote systems.

The new paradigm of Cloud computing brings about new challenges to scheduling. While the challenges from grid computing, such as scheduling and load management of distant systems still apply within cloud computing, there are added complexities that have emerged from latencies introduced by the hypervisor. The latencies are not just due to network and performance, but also from overheads that are a direct consequence of virtualisation. A finite amount of time is required to provision the required execution end point. A scheduler needs to take into account the execution end-points performance capability,

which is slowed down by hypervisor overheads, usual normal transfer overheads, and provisioning time.

3.4 Available Schedulers

There are many commercial job schedulers currently available in the market for use on HPC systems. Along with experimental and proof of concept schedulers there are several open source schedulers with a healthy development life cycle. Without going into details of each, there are some names that must be mentioned both for their importance in the market and to better understand the development route for the Mouldable Scheduler.

One of the pioneering job management softwares is the Portable Batch System (PBS), later known as openPBS (Bayucan & Henderson, 2000). This system currently exists as a pay-for-use software in the form of PBS Pro which includes both a job management system and a job scheduler. An open source variant also exists as part of the Terascale Open-Source Resource and QUEue Manager (TORQUE) suite (Computing & Computing, 2012). The Sun Grid Engine (SGE) was another commercial alternative that was widely deployed. Since the Oracle takeover of SUN, Grid Engine exists as its own project with commercial and open source variants. Since TORQUE and PBS are just job management systems or batch queuing systems they had limited scheduling capabilities. As part of the Cluster Resources offering there is Maui that is a pure scheduling software. It has the ability to integrate with the openPBS based queuing systems. Cluster Resources also offer a comparable software suite known as Moab. Moab is a commercial software that includes the functionality of both TORQUE and Maui (Jackson, Snell, & Clement, 2001).

Other popular commercial offerings are Load Leveller™and LSF™. While the former is tied to IBM systems only, LSF is becoming a popular middleware. Due to a simple licensing structure and maintenance/support contracts, many cluster Original Equipment Manufacturers (OEM)'s are supplying their systems with LSF™. The four commercial packages discussed above are now gearing towards elastic and "cloudy" environments.

HTCondor developed by the University of Wisconsin is a mature robust system for job management and scheduling in high throughput environments. Apart from the major differences introduced with regards to HTCondor being a HTC middleware, it also differs from the above mentioned HPC schedulers because its domain is fluid. This means that a HTCondor HTC system is dynamically scalable without modifications to the central manager. End point resources can connect and disconnect without disrupting the operations of the central manager. While a node can go down in an HPC environment, a node cannot join in unless it is predefined. This makes HTCondor a popular choice as a scheduler in an elastic environment (Montero, Huedo, & Llorente, 2006).

3.5 Simulators

Modelling user behaviour is key when testing the effectiveness of a system. The ability to model user behaviour is used by designers and manufactures in the product design cycle and is also an important tool for system administrators who are making changes to configurations. The users' workloads - their job submission behaviour, is an important metric for managing HPC, HTC and grid systems and for designing systems or scheduling algorithms. Most publicly available simulators are either inflexible or tied in to proprietary scheduling

systems. Academic literature relating to simulators does not link back to actual programs that can be utilised for further research.

A variety of grid simulation tools already exist. These include SimGrid (Legrand, Marchal, & Casanova, 2003), GridSim (Buyya & Murshed, 2002) and the Maui (Sterling, 2002) scheduler in simulation mode. SimGrid and GridSim are primarily focused on large geo-distributed systems. As a result these simulators are more concerned with hardware specifications and process IO interactions than the cluster schedulers.

The Maui scheduler does focus on cluster testing but does not allow for testing of algorithms which are not a part of the Maui software. The significance of this is an innate inability to test new algorithms, or alternative middleware platforms that do not use the Maui scheduler. Further as Maui was originally written to work with a multitude of batch queuing system it has its own nomaclature and formating for data input. Poor documentation, a slow down in development, and the lack of simple tools to convert logs from a batch queuing system e.g. TORQUE, means simulating workloads is non trival.

3.6 Summary

Job Management Systems include two primary components - the batch queuing system and the job schedulers. The batch queueing systems are typically mature pieces of software that generally handle the mechanics of the system. This thesis will concentrate on job schedulers and propose new paradigms to improve the efficiency of the JMS.

As presented in Section 2.7 all shared systems, which make the bulk of the new paradigms of large compute resources, require some form of system to manage jobs submitted by the users. One of the biggest challenges for system

administrators and researchers is the lack of flexible simulators that are able to give the operators some guidance as to how their actual workloads will perform under different system characteristics. In an energy concious world for example the question ”Do we really need so many nodes on at this time” is a fundamental one that cant be answered without a proper simulator.

In the next chapter (4) a detailed description of scheduling alogirthms and their associated literature will be presented. Chapter 5 will bring together the preceeding chapters, outline the Research Computing Infrastructure (RCI) present at University of Huddersfield, and the short falls in Job Management faced.

Chapter 4

Literature Review

4.1 Introduction

This section analyses the existing research in several niche subject areas. Each of these subject areas come together to inform the development of the Intelligent Robust Mouldable Scheduler. The job management systems are governed by scheduling algorithms, scheduling strategies and scheduling policies. Section 4.2 builds on the background presented in Chapter 3 and explores the different layers that govern scheduling systems; Section 4.3 looks at the role of intelligent scheduling systems in High Performance Computing (HPC); Section 4.4 covers research and development in benchmarking techniques; and finally Section 4.5 addresses a key piece of work that has inspired the research reported in this thesis.

4.2 Scheduling Techniques

There are three important paradigms of scheduling in a multi-user parallel environment that are reported in literature focusing on scheduling strategies. The three paradigms are rigid scheduling, mouldable scheduling and malleable scheduling. It is important to note that parallel environment in this case refers to a system that can run multiple processes or jobs in parallel and not the traditional meaning in HPC to imply a single job running across multiple cores. Parallel environment also differs from the concept of multi-tasking. There is a degree of time-splicing that takes place in multi-tasking which does not exist in a parallel environment. Distributed environment also differs from parallel environments. For the purposes of this report the former refers to multiple mutually exclusive compute systems. These systems could be standard multi-tasking systems, parallel systems or both. This section also covers scheduling strategies within elastic or cloud based environments.

4.2.1 Traditional Scheduling Strategies

In parallel environments a job scheduler has two parts to consider – the selection of the machine and the scheduling of the jobs over time. Within the selection strategies there are two constants: number of nodes available in the execution environment and the number of free or available nodes/end points/slots at the time of calculation. A limited list of strategies that a scheduler can follow is:

- Biggest Free, where a new job is given to that node which has the most endpoints and slots free. The drawback being that wide jobs suffer as earlier submitted narrow jobs make take vital resources

- Random, where a random machine meeting the criteria from the pool is assigned the job. Mathematically this provides the fairest distribution over time
- Best Fit, where the scheduler leaves the least amount of resources free. While this provides the best utilisation and does not raise the issues from Biggest Free, this sort of out of order execution can result in jobs being delayed indefinitely
- Equal Utilisation, where the scheduler works as a load balancer favouring least loaded machines. This too leads to vital jobs getting stuck in the queue.

(Hamscher, Schwiegelshohn, Streit, & Yahyapour, 2000)

For scheduling jobs some of the popular algorithm are:

- First Come First Served (FCFS), as the name suggests only looks at the submit time of the job. In a multi-profile job execution environment this can lead to wide jobs blocking the queue waiting for resources to become available while short narrow jobs may end up waiting a long time, resulting in a poor quality of service. In distributed compute environments if there is a centralised scheduler then this algorithm proves to be the most effective in maximising utilisation and QoS (Hamscher et al., 2000).
- Backfill, which is an out of order FCFS strategy that does not hold a queue to provide resources for wide jobs. Using running-time information the scheduler fills in gaps with short narrow jobs. This algorithm has two strategies
    1. Conservative, where the system allows jobs to run out of order as long as the expected start times of any other job do not change

2. Aggressive, where out of order execution is enabled to ensure maximum utilisation even if that correlates to a delay in other jobs. High priority jobs are given preference

These refer to the time margins the scheduler is willing to keep when attempting to maximise the utilisation. Studies have shown that Aggressive backfilling is the most effective but can result in a poorer performance, similar to the FCFS example, or in worst case scenarios when users provide incorrect execution information.

- Pre-emptive, where at submit time the scheduler decides to which resources the job will be assigned

(Hamscher et al., 2000; Feitelson, Rudolph, Schwiegelshohn, Sevcik, & Wong, 1997)

Aside from an applications profile there are also four types of job profiles.

- Rigid Jobs: The number of processors assigned to a job is specified external to the scheduler and that is not to be adjusted
- Mouldable Jobs: The number of processors assigned to a job is determined by the system scheduler using some constraints provided by the user. These are determined either at the time of submission or scheduling
- Evolving Jobs: Through different stages a job requires different resources and the scheduler makes provisions during the job run time
- Malleable Jobs: The resources assigned to a job change during execution as the schedulers requirements vary over time.

(Feitelson et al. 1997)

Because most applications cannot accommodate Malleable jobs and current production level schedulers have their own offline mechanisms to handle Evolving jobs this report will henceforth only address the requirements of Rigid and Mouldable jobs.

4.2.2 Mouldable Scheduling

Expanding on the definition of a mouldable job given in Section 4.2.1, a mouldable job is where the user provides some ”recommendations” for resources required, and possibly a deadline for the job. The scheduler then allocates resources to best decrease the turnaround time (TaT) of the job. The quantity of allocated resources is adjusted based on a constraint to help maximise the throughput. This constraint can exist in the form of a variance provided by the user or in some cases, if available, the decision can be based on the scalability profiles (like the Downey Model) of the application being run (Cirne & Berman, 2001; Trystram, 2001; Cirne & Berman, 2002; Srinivasan, Krishnamoorthy, & Sadayappan, 2003; Hungershofer, 2004; Huang, Shih, & Chung, 2009; Saule, Bozdağ, & Catalyurek, 2010; Carroll & Grosu, 2010).

These studies referenced above, have shown that in comparison to rigid jobs in a first come first served queue with backfilling (aggressive or otherwise) mouldable jobs have a lower average queued time. Under other scheduling strategies too, mouldable schedulers almost always give the best result for the contradictory demands of lowest response times and highest utilisation rates (Cirne & Berman, 2001, 2002; Srinivasan et al., 2003; Hungershofer, 2004).

Some of the ”mouldable scheduling algorithms” that have been developed include:

- Submit-time greedy: Users provide different partition sizes and expected end times. Based on the conditions of the queue, the best partition is chosen, to have the lowest possible completion time. This allocation is then fixed.

- Submit-time fair share: This is similar to the previous strategy but incorporates a fair share aspect, which prevents any user from consuming too much of the system at a job level.

- Schedule time greedy: A variant of the Submit-time greedy wherethe optimal partition is chosen at schedule time.

- Schedule time aggressive fair share: A hybrid of the Submit-time fair share and the Schedule time greedy, this algorithm makes allocation decisions at schedule time and restricts the partition size based on a fair-use policy. This algorithm however includes backfilling to further streamline the utilisation.

(Cirne & Berman, 2001, 2002; Srinivasan et al., 2003)

4.2.3 Scheduling in Elastic Environments

In elastic environments job schedulers needs to take into account two new parameters. The first parameter is that the size of its execution environment is not fixed. The Condor scheduler (discussed in detail in Section 3.4), has a variable execution domain but when an end point joins or leaves the execution environment the scheduler recalculates the order of the queue. In an elastic environment (discussed in detail in Section 2.3: Review of Elastic and Shared Resources) a scheduler has the option to scale up the system on its own accord. This has to factor in any scheduling algorithms. The second parameter

is how much can it scale? This is governed by whether the elastic environment has a limit to it – may not a plausible scenario in a public cloud but definitely a reality in a private cloud. There may be budgetary constraints limiting the elasticity of the system.

The elastic environment also introduces a new constant in the calculations. This constant is the provisioning and termination time and cost of an instance. A study on efficient scheduling, keeping these factors in mind can be found here: (Zaman & Grosu, 2011). Using the term Bag of Tasks for jobs executed under High Throughput Computing (HTC) environments work carried out at Virje University outlines scheduling under budget constraints (Oprescu & Kielmann, 2010). Similar to work carried out by (Srinivasan et al., 2003) developers have looked at Just-in-time resource elasticity for cloud applications based on the scalability of the different applications. Further, the nodes free constant tends to go unused till the scheduler hits the elastic limit of the system (Jie, Qiu, & Li, 2009).

Several studies have also been carried out attempting to address automated scheduling in timeshared parallel machines but in a majority of the cases, during a time splice, a stakeholder gets the complete system (Stoica et al., 1996; Kalé, Kumar, & DeSouza, 2002; Meredith et al., 2003; Padgett, Djemame, & Dew, 2005). Al Jahdali et al. have addressed the challenge of scheduling jobs in an elastic environment where the possibility of multi-tenancy on the execution node can exist (Aljahdali, Townend, & Xu, 2013; AlJahdali et al., 2014). Having the advantage of full access to Googles trace logs Al Jahdali et. al were able to ”benchmark” the jobs and make scheduling and allocation decisions. Since the Google dataset is homogenous for application type, the benchmarking approach is to average users requirements and the corresponding run time.

4.3 Review of AI in Scheduling

4.3.1 Intelligent Schedulers

The concept of intelligent schedulers to manage tasks is not a new one. Within the realm of HPC the word 'intelligent' is frequently used to describe features. But the reality is that there is no element of machine learning and at best these systems can be described as rigid rule based systems. There does not appear to be actual production standard deployment of such systems.

Some forms of 'intelligent' schedulers do exist for centralised grid resource brokers. These systems use real time feedback from remote systems to decide where jobs and data should migrate to. Using information of system load, system specifications, network latency and bandwidth the scheduler tries to minimise the time for the job to complete the execution (including in-flight time).

Literature relating to intelligent schedulers at cluster level usually entail power management features. These schedulers are "smart" enough to turn-off nodes when not being utilised and based on some rules only bring the nodes back up when required. In their paper Goel et al. devise a way to get a per-core estimation of power and the corresponding affect on ambient temperatures. Thus the job management system is provided with more information leading to 'intelligent' operations (Goel et al., 2010). While no mechanism is provided to feed this information back, there are power management middlewares for HPC systems that can utilise this information e.g. CLUES (De Alfonso, Caballer, & Hernández, 2010; Alvarruiz, de Alfonso, Caballer, & Hern'ndez, 2012). Similar works have been carried out by HPC vendors like Dell (Iqbal, Gupta, & Fang, 2005) and Xerox (Yao, Demers, & Shenker, 1995).

Fundamental work in designing an intelligent job management system using job performance prediction has been presented in a paper entitled Evaluation of a Workflow Scheduler Using Integrated Performance Modelling and Batch Queue Wait Time Prediction (Nurmi et al., 2006). In this paper the authors have presented a platform independent model of how an application will behave using floating point operations and memory access patterns. These values are then used to estimate how the program would perform on a given piece of hardware. Using what they describe as a Binomial Method Batch-Queue Predictor and the Downey model (Downey, 1997) their solution estimates batch queue wait times. Using this data their aim is to reduce turn around times for Workflow based job submission systems. With some success they modified the VGrADS workflow scheduler (Ramakrishnan et al., 2009).

In their paper A Dynamic Scheduler for Balancing HPC Applications, Boneti et al. propose improving application performance by balancing workloads by creating a better load balance within the system. Their solution deploys a kernel extension for Linux on an IBM Power 5 system. Their definition of an HPC application falls more on multi-threaded or single node with multi-core Message Passing Interface (MPI) applications. The adaption to the Linux kernel has allowed the system (using heuristics) to predict when an application enters a core intensive phase and the system then load balances to allow for maximum Central Processing Unit (CPU) efficiency (Boneti, Gioiosa, Cazorla, & Valero, 2008). While this is a good solution it does not scale to a Distributed Multicore environment as the region of a Linux kernel does not extend beyond the node it operates on.

Pandey et al. in 2010 proposed a heuristic scheduling workflow for applications in the cloud. Based on Particle Swarm Optimisation, the proposed system selects and schedules applications in cloud computing environments by optimising best resource selection taking into account both computation cost and

data transmission cost (Pandey, Wu, Guru, & Buyya, 2010).

Within grid environments, genetic algorithms utilising heuristic algorithms have been proposed to address scheduling in directed acyclic graph (DAG) environments. DAG represents the precedence relations of the tasks of a parallel program in a distributed computing system (Woo, Yang, Kim, & Han, 1997). The Ianos middleware, developed in grids utilises an intelligent scheduling system along with Viola the grid meta-scheduling system (Rasheed et al., 2007). Khalid et al. and Stumm have done similar work of intelligently scheduling within scientific grid utilising meta-data. The former concentrated on the spawning of Virtual machines within the cluster while the later addressed decentralised scheduling (Khalid et al., 2009; Stumm, 1988).

In their paper ”Fuzzy Logic Based QoS Optimization Mechanism for Service Composition”, Avila and Djemame present an adaptation approach that implements self-optimization based on fuzzy logic. When allocating web-services in elastic environments the decision of scaling up and the possible service provisioning end points are decided based on costs, availability, historic analysis and benefit of adaptation. This approach allows then to use linguistic variables to make smarter decisions. Their experimentation has shown an improvement in the global QoS and reductions up to 17.1% in response time, 17.38% in cost and 40% in energy consumption (de Gyves Avila & Djemame, 2013).

4.3.2 Heuristics in HPC

In this section, we look at literature that uses heuristic information to improved scheduler performance. While there is a limited amount of literature on the application of heuristics in HPC environments, there are several approaches to scheduling in grids.

Mönch et al. have used heuristics to improve the scheduling of batch machines in the manufacturing process. The challenges cited in this paper are similar to those faced in scheduling on HPC systems. Within their manufacturing process there are different types of job families which have unequal ready times and unequal processing times. The most important criterion is to maximise customer satisfaction. Using genetic algorithms and heuristics, the paper presents two options for scheduling and assesses their effectiveness. In one approach jobs are made into batches. These batches are then assigned to machines and processed. In the second approach jobs are signed to machines and then batches are formed before processing. In most test cases the first approach always performs better. This result is to be expected as it allows the scheduler hindsight. Related to HPC scheduling, their approach will be effective in scheduling jobs that are already in a queue. In HPC application arrival times for jobs are not fixed therefore batches cannot be pre-made. In the manufacturing process there is a limited number of systems that perform tasks, therefore it is easier to predict runtimes. However, the application of heuristics in HPC will over time make predictions more accurate (Mönch, Balasubramanian, Fowler, & Pfund, 2005).

Within grid environments the heuristic information collated relates to time-of-flight of data and rough estimates of processing requirements and application constraints. This information helps to decide the optimum system to allocate to the job (Weng & Lu, 2005; Agarwala, Poellabauer, Kong, Schwan, & Wolf, 2003; Schwan et al., 2005). In both cases the quality of service or turn around time of the job improves. Weng et al. propose an algorithm that copes better with increasing input data size.

4.4 Benchmarking Schemes and Schemas

Benchmarking is an important facet of computer science. Following Moore's Law, computer architecture is evolving rapidly. Benchmarking schemes are the main tool to gauge how effective these new developments are. Microprocessor manufacturers have used benchmarking schemes to create hype in the market and assist sales. Review magazines also utilise benchmarking schemes to ensure consumers are better informed.

LINear equations software PACKage (LINPACK) (J. J. Dongarra, Bunch, Moler, & Stewart, 1979) is one such benchmarking scheme that has been popular for several decades. LINPACK uses linear algebra based FORTRAN problems to calculate the Floating Point Operations Per Second (FLOPS) rating for a given system. While LINPACK has been succeeded by LAPACK as a benchmarking tool for shared memory and vector processor systems, High Performance Linpack (HPL) is still the standard used to benchmark the fastest computers in the world for the Top500 list (Petitet, 2004).

LINPACK and other benchmarking tools like it are classed as kernel benchmarks (Sayeed et al., 2008). Kernel Benchmarks are usually specialised towards measuring individual system components. These benchmarks are able to give peak performance of the various components – though this is always assumed to be under ideal conditions. Kernel Benchmarks are unable to give an overall performance metric which can be used to fine-tune the system. Even by combining various kernel metrics together we cannot predict the performance improvement of one benchmark for a small change in another (Sayeed et al., 2008).

For system administrators developing a new system, kernel benchmarks are

still effective in giving developers figures to improve upon. This sort of implementation has also led to tools to provide good analysis of CPU and system utilisation leading to better efficiency (Said, Taib, & Yahya, 2008).

Another aspect of evaluating machines is to attempt a performance prediction for a family of code. These predictions are based on the scalability models for the code. Efforts by Allen B. Downey (Downey, 1997), outlined in a report titled ”A model for speedup of parallel programs” have led to the establishment of what others have referred to as the Downey Model (Srinivasan et al., 2003). The Downey model establishes ”a family of speedup curves that are parameterised by the average parallelism of a program, A, and the variance in parallelism” (Downey, 1997). The performance characteristic of a particular algorithm, like a CFD code, is modelled to predict the run time if the cluster size is changed but the dataset it kept the same. This model utilises observable and measurable program characteristics, such as run times for a particular section of code and then creates an average. However, at times it falls short when attempting to calculate averages. If the timings are linear or near-linear on any range of system size, for the particular dataset size, the model cannot calculate an average. There is no guarantee that at a different dataset size the algorithm keeps the same speed up profile.

Further efforts, like that done by the San Diego Supercomputer Center [sic], attempt to create a multidimensional benchmarking scheme which provides more information than Cycle-accurate models (similar to the Downey model only more elaborate and detailed) and run times from real applications. Their approach ”attempts to see how much of the factors that affect performance can be attributed to few parameters only adding complexity as needed to explain observed phenomena.” This approach sits between kernel benchmarks and real world application testing. The authors are averse to benchmarking schemes that collect many kernel benchmarks and algorithm specific characteristics to generate a

performance model. They have found that this, very accurate method, is not feasible to model full application packages on large scale systems due to time and monetary considerations (Snavely et al., 2002).

As mentioned before Benchmarking Schemes, especially kernel benchmarks are mostly utilised by manufacturers to sell their products (Sayeed et al., 2008). For many users of high end HPC systems predicting how their code or application will run is of more pressing concern. As the Downey model laid the foundations for analysing the scalability of algorithms other research and development has been carried out to better model multidisciplinary software (Naik, 1992; Kerbyson et al., 2001; Amit, Caspi, Vitale, & Pinhas, 2006; Elton et al., 2009). This does not however answer the question ”If I buy this system how fast will my particular code for my particular problem run?”

The natural evolution from kernel benchmarks and then algorithm performance modelling is real world application benchmarks. In their paper ”Measuring High-Performance Computing with Real Applications” Sayeed et al. make a case for the development of a benchmark that uses actual applications to grade systems. The paper is aimed at establishing an application based benchmark to assess performance of machines to influence ”buying time” decisions. Aside from the political and financial factors limiting the development of real world application benchmarks, there are also technical limitations. Benchmarks discussed above aim to provide developers with figures to help steer improvement of the hardware infrastructure. Real Applications will not be able to help with this. Fine tuning components in a system to improve the performance of a particular application is possible, but if the system runs more than one application then this approach will lead to a fall in service quality. For manufacturers and hardware designers real applications are not the best yardstick as ”today’s real applications might not be tomorrow’s.” Propriety applications also cannot be fairly utilised as manufacturers and developers can work together thus tainting

the results. Real application benchmarks, barring one use case, appear to be in the same category as its predecessors like LINPACK – just marketing devices sometimes leading to fine tuning monitoring and measuring devices (Sayeed et al., 2008).

Work carried out by Simon et al. and Alam et al. creates a foundation for benchmarking HPC Systems and new generation processors with real world applications. Their work outlines how best to use such benchmarks to assess the new hardware effectively before deployment. Alam et al. work also includes bringing together kernel benchmarks and application specific benchmarking to create a complete picture of a systems performance (Alam, Barrett, Kuehn, Roth, & Vetter, 2006; Simon, Cable, & Mahmoodi, 2007).

For the purpose of assisting in new purchases, real application benchmarks can be very effective. Our interest in real application benchmarks stems from the question ”For a given application on a given system is it possible to create a metric of performance versus dataset and problem size?” Benchmarking suites like Perfect Benchmarks (Cybenko, Kipp, Pointer, & Kuck, 1990) and the Stanford Performance Evaluation Corporation (SPEC) (Eigenmann & Hassanzadeh, 1996) are two real application benchmarks that have withstood the test of time and are still up to date, unlike many others from that era (Sayeed et al., 2008). The SPEC benchmarks are a compilation of popular applications from each domain of scientific research with associated workloads and datasets. Further work has been carried out to include some level of kernel benchmarks, though these are limited to the application. When benchmarking, along with the turn-around-time (TaT) gathered by SPEC, add-on components also provide information about the inter-processor communication, instructions and FLOPS per cycle and the I/O calls and performance. Thus meeting some of the shortfalls outlined above (Eigenmann & Hassanzadeh, 1996; Sayeed et al., 2008).

Through experimentation with the above mentioned benchmarking suites it is felt that while the benchmarks are able to provide information about an applications performance these benchmarks are lacking an ”open-framework” to allow a user to put in his/her own workload and dataset into the mix. The benchmarks, probably due to their maturity, are rigid in nature.

If adjusted, SPEC or Perfect Benchmark with their associated add-ons can be used to feed information back to a running system to help improve the load. Work has also been carried out to benchmark and predict the scalability of embarrassingly parallel or high throughput applications and workloads (da Silva & Senger, 2010; Montero et al., 2006). Together these benchmarks will provide better insight of the workload running on the University of Huddersfield Research Computing Systems (RCS), the Queensgate Grid (QGG).

As mentioned above, adjusting the benchmarks to feed information back to the system would be a useful addition to help the system manage its workload. Work has been carried out to develop a mechanism for interpretive performance prediction (Parashar & Hariri, 1997). Using a modular approach this suite uses a ”System Module” which characterises the hardware, an ”Application Module” which incorporates the application characterisation methodology, an ”Interpretation Engine” (module) which derives the execution costs and requirements and the ”Output Module” which creates the performance metrics and returns the results to the user. While the output from the previous example is geared towards high performance application development, a similar approach to work with the above benchmarks can lead to an innovative and useful product.

4.5 Seminal Works

This thesis takes inspiration from work being carried out at the Ohio State University by Srinivasan et al. Their paper ”A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs” by Sudha Srinivasan, Savitha Krishnamoorthy and P. Sadayappan (2003) has strongly influenced the system design and testing approach adopted in this thesis.

Where most algorithms rely on users giving a recommendation on partition size and will then choose the best option at submit or schedule time, Srinivasan et al.'s approach includes algorithm scalability characteristics to further improve the allocation decision. Adopting the Downey model as a framework Srinivasan et al. are able to prove that letting the scheduler decide the variance, and then allocate resources helps to improve turn-around times for jobs in a FCFS queue with aggressive backfilling and a Fair Use policy. The results also show improved usage efficiency and more robustness on the part of the system.

The initial premise to their paper is that up to that point mouldable algorithms were still *rigid* as they were dependant on user input and didn't take into account application performance characteristics. Using real trace logs of a small system at the San Diego Computing Centre and NAS benchmarks for their running algorithms Srinivasan et al. are able to create a mouldable workload. Working backwards, the execution time in the trace is matched to the results of the NAS benchmark for the application. This approach gives the authors a point on a scalability curve derived by the Downey model for the application. This point on the line uses the following information: application, number of cores requested and the execution time. The Downey model is then employed to calculate execution times at other node allocations.

With this scalability information and use of the aggressive backfilling algorithm Srinivasan et al. are able to ensure improved performance across all classes of jobs. The class divisions are created along the lines of workload-weight. Workload weights are calculated using number of cores requested and execution time. The authors maintain that FCFS systems cause fragmentation and under utilisation and the introduction of backfilling in mouldable scheduling leads to poor turn around time for some workload-weights (it is assumed they mean despite mouldable scheduling large jobs get delayed, but they do not explicitly state this) (Srinivasan et al., 2003).

Srinivasan et al.'s work signals a step change in thinking when it comes to mouldable scheduling. Taking out the *rigidity* within existing mouldable scheduling approaches by allowing the system to make allocation decisions using application performance characteristics is a major step towards autonomic workload management. While they did not emphasise it, their approach to creating synthetic logs to test mouldable schedulers is novel and very important. Other papers around the same time have not strongly documented how they have created their workloads or how their workloads correlate to real observable workloads on HPC systems.

However there are some assumptions made which lead to shortfalls in the approach. Primarily it is the use of the Downey Model to evaluate scalability of workloads. The Downey model is a system agnostic approach to determine application scalability. The Downey model does not take into account network speeds and overheads (e.g. in clusters), location of the data (e.g. in clusters and clouds), or multi-tenancy nodes (e.g. in clouds). Even though the authors used the NAS benchmarks, the results are not used to find the true scalability across the range of allocations. They use the user specified core allocations in the trace logs to make assumptions about both the size of the dataset and

the workload information. Observation of workloads on the University of Huddersfield Queensgate Grid reveals allocations of 2 or 4 nodes are very common and cover a large range of dataset size[1]. However the execution time of two jobs using the same application can be in the same range but have different workload and dataset sizes. This is why Srinivasan et al. assumptions seem to introduce larger errors into their testing.

Srinivasan et al. also do not take into account queue wait times in their system workload. This over sight is further compounded by the fact that the authors assumption of under-utilisation in a FCFS scheduling strategy refers to assertions made by Krallmann et al. in their 1999 paper entitle ”On the Design and Evaluation of Job Scheduling Algorithms”. This paper does not take in account mouldable scheduling, and only tests and refers to rigid workloads where only a single *user specified* allocation can be made for any job (Krallmann, Schwiegelshohn, & Yahyapour, 1999).

Under a mouldable environment the scheduling algorithm *should* be able to fit jobs into vacant spaces if the turn around time can be improved. A mouldable scheduler needs to take into account the queue time of a job into the ”total time” for the job. Based on this and the ability to mould a job, the chances of large swathes of the system sitting idle are very small. The exception would usually occur if the system assumes the minimum allocatable node count is greater than 50% of the system. In a heterogeneous user system the introduction of backfilling will also remove the small percentages of idle system.

[1] Section 6.3.2 shows how many classes of datasets for one application and one workload share this allocation range

4.6 Summary

This chapter presented an outline of the various scheduling algorithms, policies and strategies. At a fundamental level, there are scheduling algorithms that can be applied to many scheduling strategies. This thesis is focused on a mouldable scheduling strategy and will take into account First Come First Served and First Come First Served with Backfilling scheduling algorithms.

As efforts to make large, complex and more autonomous schedulers are ongoing, the use of Artificial Intelligence (A.I.) techniques like fuzzy logic, heuristics and machine learning are being adopted. These improve the overall scheduling performance of HPC systems with minimal operator input.

Thus far this thesis has outlined the key characteristics of RCS, Job Management Systems, and Scheduling techniques. The next Chapter will explore the University of Huddersfield research computing environment and analyse the typical users and workloads handled.

Chapter 5

University of Huddersfield Research Computing Infrastructure

5.1 Introduction

Since 2009 the University of Huddersfield has invested heavily in developing its in-house Research Computing Infrastructure (RCI). As the University has moved towards being more research led, giving researchers access to Research Computing Systems (RCS) type machines became essential. Within the University environment the computer clusters and other systems had to support a wide array of applications. The roadmap for research computing at the University is set by the High Performance Computing Research Group (HPC-RG), which is a group of academics from different disciplines and members of the University's IT services. The day-to-day management of the RCI is handled by the High Performance Computing Resource Centre (HPC-RC).

The different systems that have been deployed to support research computing along with the software stack and supported applications are outlined in Section 5.2. The profile of users and the systems utilisation is discussed in Section 5.3.

FIGURE 5.1: The Beowulf cluster Eridani: Cold Isle

The need to provide Quality of Service (QoS) to our users and improve resource utilisation has lead to the development of a new mouldable job scheduler.

To evaluate the mouldable scheduler, and the intelligent mouldable scheduler, described in detail in Section 8 and 9.2 respectively, a snapshot of a real workload from an existing system is utilised. This snapshot forms the input for the simulations when the tool is tested and evaluated. The Workload Sample (described in Section 5.4) evaluates, in detail, resource utilisation. Job arrival and completion rates are presented to illustrate QoS that is inherent in the system.

5.2 Research Computing Infrastructure

5.2.1 Systems

In an effort to become a world leading research institution the University recognised the importance of High Performance Computing (HPC) for research and deployed several HPC systems. HPC systems are the most effective in processing large computational problems. The University's two primary HPC systems are known as Eridani and Sol. All systems on the campus are linked together to form the Queensgate Grid (QGG) (Holmes & Kureshi, 2010).

Eridani, shown in Figure 5.1 and 5.2 is a Beowulf type HPC cluster. It is capable of delivering a peak of 500 Giga FLOPS (GFLOPS) of computational power with a sustained power rating of 380 GFLOPS. It comprises of thirty six commodity workstations linked together over a gigabit backplane. Each node has

FIGURE 5.2: Networking and Power for Eridani: Hot Isle

FIGURE 5.3: SOL Cluster in the University Datacentre

a 4 core Intel ® processor. Entire system control, monitoring, Message Passing Interface (MPI), and user data is delivered over the single gigabit network channel. The nodes are housed in a custom build shelf. The system uses Community Enterprise Operating System (CentOS) version 5 as the operating system with Open Source Cluster Application Resources (OSCAR) 5.1b2 providing the linking middleware. Terascale Open-Source Resource and QUEue Manager (TORQUE) 2.5.7 with Maui 2 manage the system resources and jobs. Users home storage is mounted from a mirrored Gluster File System Storage Server.

The second HPC cluster Sol is a more tightly coupled system. It consists of 64

compute nodes that are based on SUN X4200 server hardware with two dual core AMD® Opteron™ processors each. The system is housed in two server racks (as shown in figure 5.3). There is a separate gigabit network layer for each of monitoring and control; user data; and interprocessor MPI. Each rack, which has 32 nodes, has three of its own switches to deliver the different network layers. Each switch is linked across each rack using an ring configuration giving a throughput of 80 gbps. Sol runs CentOS 6 as the operating system, with Warewulf Cluster Manager as the middleware. Jobs and resources are managed by TORQUE 4 and Maui 3.3.0. Despite the hardware being older, the additional network layers and the latest software has ensured that Sol is the most powerful system available on-campus. Using High Performance Linpack (HPL) Sol achieved a sustained performance rating of 1.03 Tera FLOPS (TFLOPS) with a peak of 1.9 TFLOPS.

In addition to two HPC clusters, there are two specialised HPC systems that make up the Huddersfield RCI. The first is an Nvidia® and Intel® based Graphic Processing Unit (GPU) cluster. This cluster comprises of two compute nodes and a head node linked via Gigabit Ethernet. The nodes connect to the GPUs, housed in a special chassis, using a Peripheral Component Interconnect - Express (PCI-E) interface. This system runs Microsoft® Windows HPC Server 2008™ as its operating system and middleware.

The second system is data mining cluster made up of 14 nodes. This cluster, QGG-Hadoop, uses CentOS 6 as its operating system and Apache's Hadoop software as the job manager. This system is utilised to tackle big data problems, like error detection in medical records.

In order to deliver the latest in HPC technology to the users the University of Huddersfield has bought into a shared HPC service. This system is housed at the Science and Technology Facilities Council (STFC) Daresbury. The system

has been listed on the Top500 list since 2012. The University of Huddersfield pays for priority access to a set amount of CPU hours each month. Once the CPU quota is exceeded, jobs initiated by the Huddersfield group on the system become the lowest priority.

The University of Huddersfield campus network is also linked by several High Throughput Computing (HTC) middleware. This allows the HPC-RC to farm jobs to idle CPUs on campus. The primary HTC system is known as QGG-Condor. QGG-Condor is a heterogeneous pool of resources linked using HT-Condor. All the major schools of the University share their systems by deploying the HTCondor client on their lab machines. This has led to the creation of a computing network with over 5000 processing slots. The engineering department runs virtualised Linux environments that HTCondor can utilise when idle. Even at the absolute peak hours of term, it has been observed that there are at least 500+ slots available for use (Gubb, Holmes, Kureshi, Liang, & James, 2012).

The School of Arts does not run HTCondor within their labs. To meet their computational needs a second high throughput middleware Autodesk ® Backburner™ is utilised to harness idle CPUs. Backburner is a proprietary middleware that can only be used by Autodesk software. Within the University of Huddersfield it is primarily used for rendering graphics from 3D Studio Max.

All systems on the Huddersfield Queensgate campus are linked by multiple grid interfaces. This allows users to stay with the Job Description Language (JDL) they are most comfortable using. Primarily, the campus grid is a single sign-on network with a globally shared file system. All users must connect and authenticate to the central access server named Bellatrix. This system is connected by very high speed to the University backbone and to a storage server. Users can seamlessly SSH to any computing end-point and submit natively. This creates a

”trusted grid” environment. The links are established over a private high speed fibre optic network that is dedicated to the HPC environment.

The more traditional form of the grid is delivered using the Globus middleware. From any of the compute end-points or the central access node, a user can invoke processes on remote end-points using the Globus toolkit. Each system on the QGG trusts a Huddersfield only Virtual Organisation (VO). Using the Globus JDL a user can submit jobs to the HTCondor or the HPC clusters. Users however need to know the location of the required applications, that is, which system the application is installed on and where on the system the binaries exist.

Those users familiar with HTCondor can use the Condor-G features to submit jobs from the HTCondor node to the other HPC clusters on the campus network. While HTCondor is configured to ”know” about all the HPC resources in the network, users still need to know the location of their applications. Building on top of the Globus and HTCondor installations is the gLite middleware. Through gLite users get the advantage of a single sign-on, single JDL, and access to all the applications and execution end-points without having to explicitly know about them (J. Brennan et al., 2013). This system has its benefits and drawbacks. Debugging errors becomes more cumbersome as there are now multiple points of failure. However the gLite system can lead to a degree of load balancing, as gLite will divide jobs across different systems if the application is available. A second advantage is that users can use the gLite system to seamlessly scale beyond the campus grid to the UK national grid and the European grid.

The University of Huddersfield was an affiliate member of the now defunct UK National Grid Infrastructure for eScience. Local compute resources at Huddersfield were linked to the national gLite deployment for use by researchers at

other institutions. Several institutions still share resources for training purposes.

5.2.2 Applications

The QGG is regularly used by researchers from 7 departments in 3 schools. The suite of applications installed globally on the QGG are a reflection of this diverse group of researchers. Each application has a different performance profile and execution behaviour. Some applications that make up the ”usual” workload are described below.

ANSYS®Fluent This software package is the mechanical engineers primary tool on the QGG. Fluent is one of the Computational Fluid Dynamics (CFD) components that make up the ANSYS engineering software suite. A CFD simulation is an evaluation of fluid flow problems using numerical methods. Scientists can use highly accurate models within the CFD packages by using Computer Aided Design (CAD) software packages to create the object geometry. The use of HPC systems has enabled these complex geometries to be evaluated against various numerical algorithms which was not possible using standard workstations. While Fluent incorporates many common algorithms to solve a problem domain, ANSYS boasts of algorithm improvements and optimisations in their solvers. Since the software is proprietary and closed source it is difficult to judge the true appearance of the solvers.

DL_POLY is a general purpose serial and parallel molecular dynamics simulation package developed at Daresbury Laboratory by W. Smith, T.R. Forester and I.T. Todorov. The original package was developed by the Molecular Simulation Group (now part of the Computational Chemistry Group, MSG) at Daresbury Laboratory funded by the Engineering and Physical Sciences Research Council (EPSRC) and supported by CCP5. Later developments were also supported by

	1node	**2nodes**	**3nodes**	**4nodes**	**5-7nodes**	**>=8nodes**
Daily(jobs)	13	5	0.6	1	0.5	0.5
Weekly(jobs)	63	35	3	6	2	2
Monthly(jobs)	280	155	13	26	7	9
Duration(hr)	8	12	2	16	11	14

TABLE 5.1: Average breakdown of Jobs on the Eridani Cluster

the Natural Environment Research Council through the eMinerals project. The package is the property of the Central Laboratory of the Research Councils.

5.3 System Usage

The Eridani Cluster has been operational since 2011. It serves as the stepping stone for all new users of HPC/HTC technologies. It is used as a teaching tool for students studying HPC technologies and those who need HPC for their discipline (e.g. mechanical engineers). The system is also utilised by researchers (staff and students) from different disciplines. Eridani serves as the training and simulation platform for early to mid-career PhD students. While researchers have small or developing data sets Eridani is the preferred platform. It also serves as a platform for post-processing.

Due to the University of Huddersfield's lab computing environment being predominantly Microsoft Window based, the High Throughput Condor (HTCondor) service (particularly in its initial days) did not meet the needs of many users. Since many codes emerging from the Daresbury Laboratories and CERN are *nix based, Eridani fields many serial/high throughput jobs. Isolating those jobs requiring parallel libraries job submission patterns for parallel loads can be deduced. Table 5.1 shows how resource requests are broken down on the Eridani cluster on a weekly and monthly basis. The average duration of these jobs is also shown in Table 5.1.

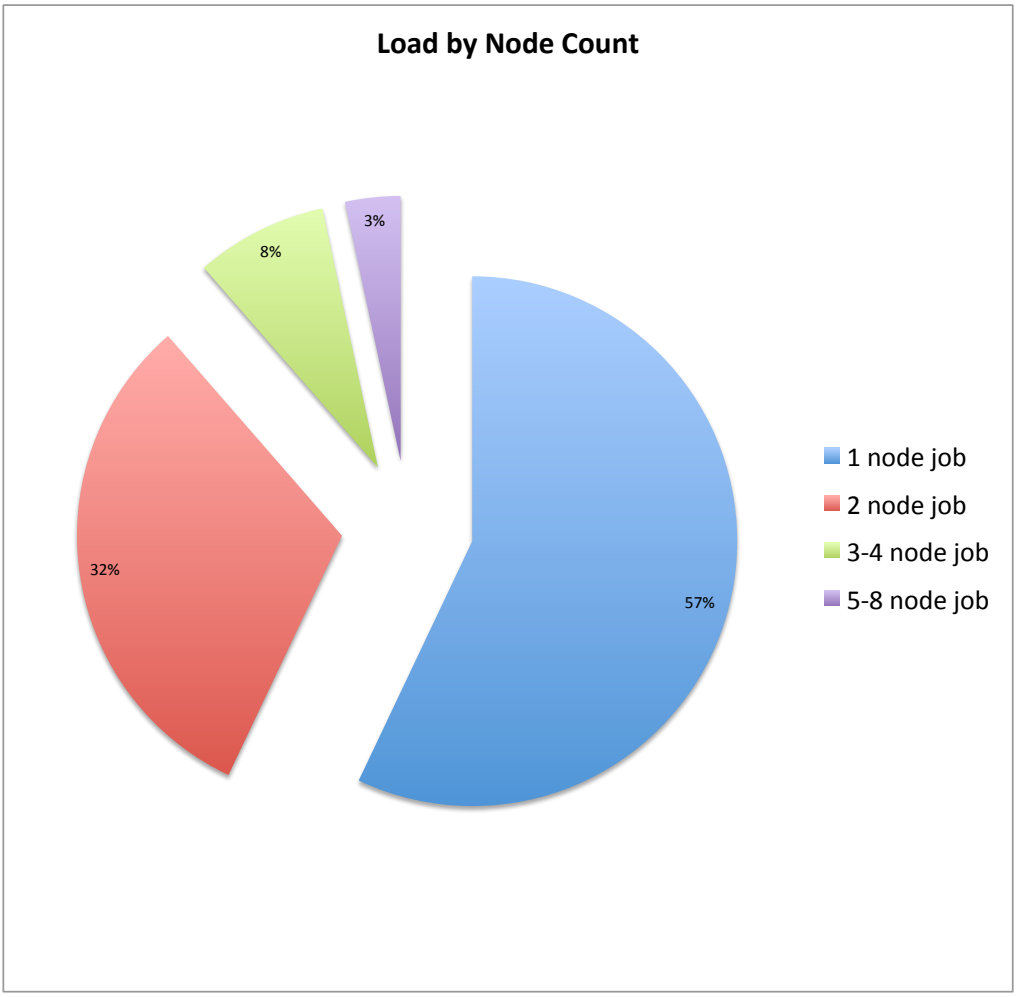

FIGURE 5.4: Load on Eridani Cluster by Node Count

These statistics are based on three years of usage logs of the Eridani Cluster (2011-2013). During this time a total of 112,390 jobs were executed amounting to 439,210 hours of computing. Ignoring the serial jobs or single core jobs we can see that there are more single node jobs than any other configuration (57%). But multi-node jobs did account for a majority of the total time spent computing (52%). Figures 5.4 and 5.5 present a node level break down of job requests over the three years.

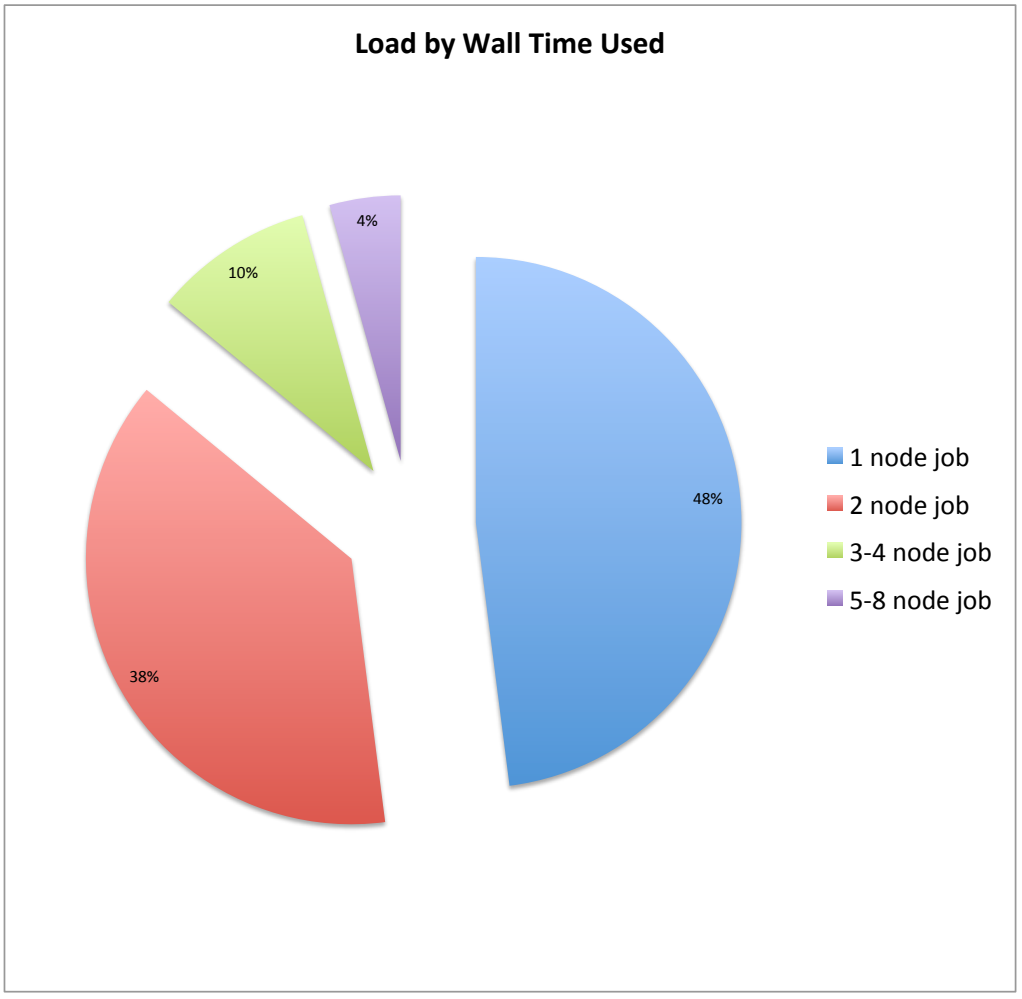

FIGURE 5.5: Load on Eridani Cluster by Job Duration

5.4 Workload Sample

To test the mouldable scheduler the month of April 2013 was identified as an ideal case study month. April is typically a busy time for the system. There are Conference and Camera Ready papers, assignments, and Research Progression deadlines in May/June. Unlike large national systems or tightly coupled large parallel systems, the load on University of Huddersfield systems varies. This variation is predicable and is quite often seasonal. The month of April 2013 is also significant as it saw three power users and one new user heavily use the system. The users were equally split between Engineering (using a CFD package) and Applied Sciences (using a Molecular Dynamics (MD) package). Using

User	Total Jobs	Average #Nodes	Max #Nodes	Average Duration
CFD-User1	226	2	4	24hr
CFD-User2	80	3	3	13hr
MD-User1	703	2	4	16hr
MD-User2	141	4	8	12hr

TABLE 5.2: Breakdown of Jobs by Power Users in the month of April

the system logs and figures from Ganglia (the Eridani system monitoring tool), the occupancy rate of the cluster for the whole month has been identified as 99-100%.

During this period the identified users submitted a total of 1150 parallel jobs. Outlined in Table 5.2 is an anonymised breakdown of jobs submitted to the system by users. Using a data mining techniques (via Apache Hadoop) the average job arrival rate and completion rate for this time period have been identified. This is discussed further in Section 7.5.

This submission pattern and workload forms the basis for the specific testing in further chapters.

The Eridani cluster is geared to handle core-serial, node-serial and multi-node jobs. Therefore not all jobs are mouldable. Due to the specification of other clusters within the University of Huddersfield RCI the workloads on the other cluster are 100% mouldable. If a job can scale across nodes using MPI and the data does not need to be pre-partitioned then it can be considered a mouldable job. Standard practices within MPI based programming mean that by and large most applications that use multi-node communications are mouldable.

5.5 Summary

In this chapter, the computing environment at the University of Huddersfield has been outlined. The typical applications and the typical load characteristics have been presented. From these statistics, 5 users and their workloads have been identified as quantifying the ”average load” on the Eridani cluster. As the mouldable scheduler is designed, to ensure accuracy, these users submission rates and completion rates were monitored.

Chapter 6

Application and System Performance Profiling

6.1 Introduction

In order to manage resources and queues within an High Performance Computing (HPC) system it is essential to know how an application will perform. As observed in Sections 1.1, 2.2 and 3.2 this information is typically taken from the user in the form of a job property specifying the 'maximum required time required'. Job schedulers also rely on the users prescribing the required resources (nodes/cores or memory).

This method has proved to include a lot of 'bad-put' and inevitably the system can not be optimised beyond a point. As discussed in Sections 1.1 and 4.5 this bad-put can either be resource under-booking leading to system crashes making the system less robust. Alternatively resources are more often than not over-booked (Chen, Lu, & Pattabiraman, 2014), wasting resources (computational and electrical/cooling) and reducing the overall Quality of Service (QoS) delivered to the users.

The work carried out for this thesis aimed to explain novel approaches to automating resource allocation in order to facilitate better utilisation of HPC resources. To be able to autonomically managed the job occupancy within HPC system it is critical to predict application run times, optimum resource allocations, and minimum resource allocations. As has been observed in Section 4.2.2 many solutions utilise the Downey model. The limitation of this system agnostic approach, especially when dealing with proprietary closed source software has been explained in 4.4. The best approach to get as close to accurate, the applications performance characteristics, is to benchmark the application on the system against realistic workloads and datasets[1].

This chapter explains the approach adopted to benchmark and collect the application performance data and how this data can be fed back to the system for mouldable scheduling (explained in Chapter 8). The end product can be utilised as a stand alone performance profiler. The code base is - Application and System Performance Profiler. This chapter is divided into two parts. The first Section 6.2 explains the development process, including design methodology, implementation and data processing. The second Section 6.3 explains the testing platform; testing applications, workloads and datasets; and the results of the application profiling.

6.2 Development of the Toolkit

When devising the Application and System Performance Profiler (ASPP) the goal was to create an open-framework benchmarking toolkit implemented in Python and MySQL. This toolkit should be fully customisable and adaptable to the system on which it is being deployed. The benchmarking categories,

[1]This work has been partially published at the 5th Balkan Conference on Informatics (Kureshi, Holmes, & Cooke, 2012)

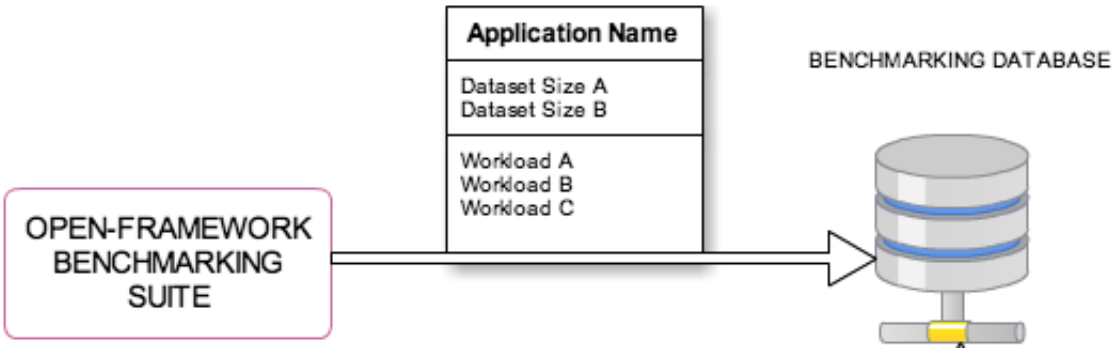

FIGURE 6.1: Benchmarking Suite

datasets, system size etc, all should be defined by the end user (who in this case is the system administrator) as shown in Figure 6.1.

The toolkit is not focused on presenting the maximum power that can be squeezed out of the system but how an application would behave with different resource allocations passed to it. This open-framework has been adopted to ensure reusability of the software external to the Mouldable Scheduler project.

6.2.1 Toolkit Architecture

As seen in Figure 6.2, the benchmarking suite works in two phases. Phase 1 is the pre-processing stage and Phase 2 is the post-processing stage. The actual 'processing' stage is performed when the datasets are running on the system and these are external to the actual benchmarking suite.

Initially, a system administrator starts the profiler with a new application and dataset combination. These are defined in a benchmark config file (see Figure 6.3). Each set of benchmarks needs a new configuration file. The toolkit uses the configuration file-name as the name for the set of benchmarks. Within the configuration file the administrator needs to define the application being used, the locations of the sample datasets and some worded classification of what

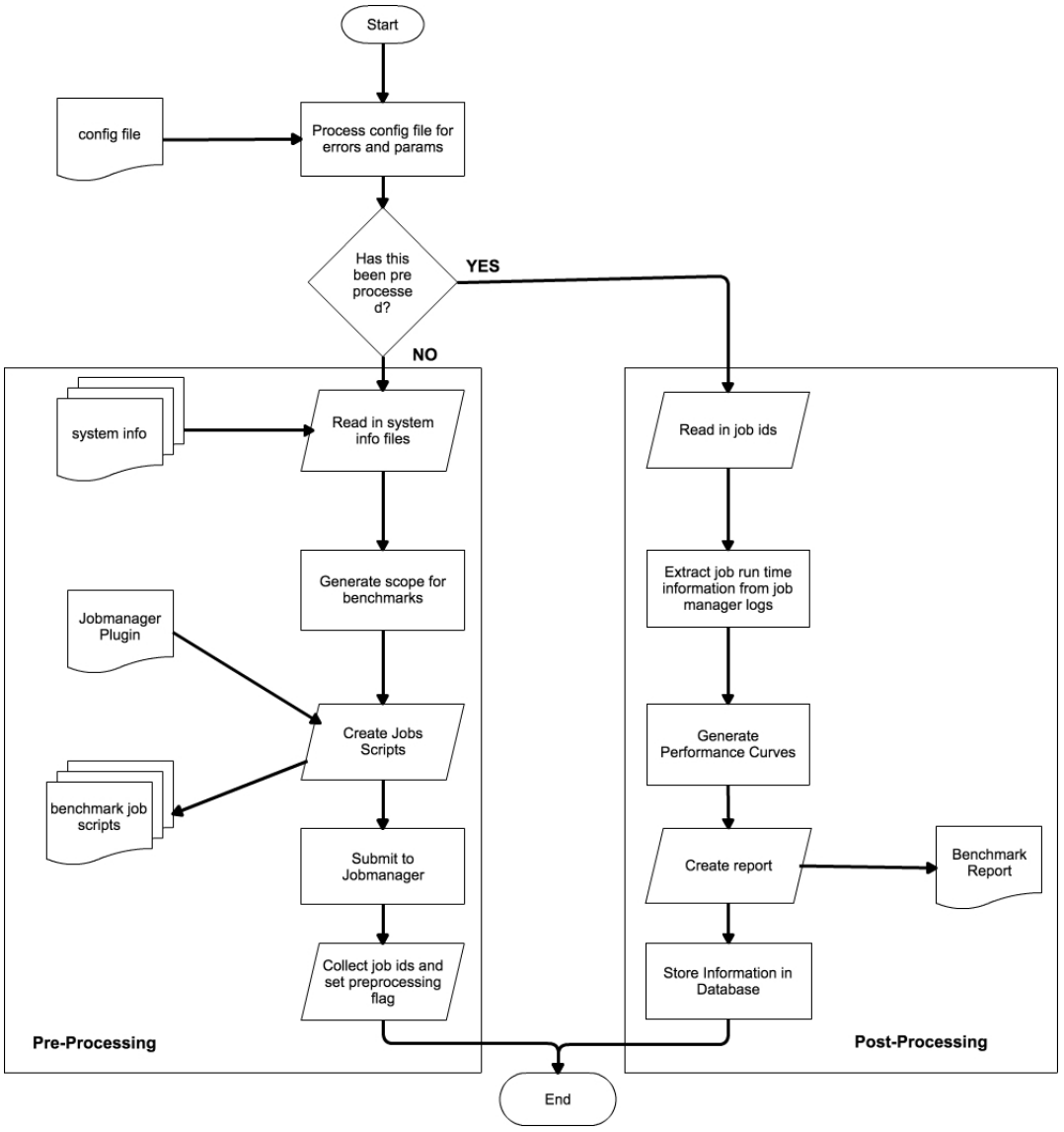

FIGURE 6.2: Flowchart for the Benchmarking Suite

each dataset is i.e. SMALL, MEDIUM, LARGE. The configuration file also includes meta information about the dataset and workflow, e.g. dataset could be in terms of molecules or elements while workload could be in terms of time or iterations. This way a mechanical engineer can query the profiler about the performance profile for his/her application in terms of elements and iterations (keywords native to CFD softwares).

The toolkit then ensures that the config file is error free before it imports system related information that the benchmarking suite will use. One such import is the system information configuration file. This file contains the system size

```
[WORKLOADS]
number-of-workloads: 2
workload-meta: iteration
workload-1: 5000
workload-2: 10000

[MODELS]
number-of-models: 3
model-meta: elements

model-1: 1014429
class-1: small

model-2: 2193408
class-2: medium

model-3: 4093656
class-3: large

[BENCHMARK_INPUTS]
small-filename 1014429.msh
medium-filename 2193408.msh
large-filename 4093656.msh
```

FIGURE 6.3: Extract from Sample Application Configuration File

information and the range of resources that are associated to the various classifications of job sizes (for example SMALL 1-2 nodes, MEDIUM 2-3 nodes, LARGE 4 nodes etc). The next configuration file that is imported is the application information configuration file. This contains command line instructions and environment variables required to start the application. The third configuration file is the plugin for the underlying batch system. The ASPP then verifies that the input datasets defined in the benchmark configuration file do exist.

To keep the Application and System Performance Profiler as flexible as possible, it has been created to be very modular. Ideally, it should be able to interface with any batch system by the use of a plugin. As shown in Figure 6.2 the third configuration file is the plugin. For the purposes of this research, efforts have been limited to PBS/Torque (Computing & Computing, 2012) based batch queuing systems but additional plug-ins for SGE and LSF are possible.

6.2.2 Generating Benchmarks

The ASPP aims to provide benchmarking information that is characteristic of normal workloads for the system. Therefore it requires user specified input files with classifications. By taking the user specified ranges of resources for each

classification, the ASPP generates several benchmark jobs. The aim is that for every workload and dataset size, the ASPP should be able to get 3 run times on different hardware allocations. The ASPP generates at least 3 benchmark jobs for every dataset and workload size within the range of its classification. For example, a CFD application with 2M+ elements and a workload of 1000 iterations can be defined as a medium workload so the ASPP will generate 3-4 jobs (the more points the better) which will run the simulation at 1 node (4 cores), 2 nodes, 3 nodes and 4 nodes.

The aim is to generate a curve of *wall-time* versus *nodes*. Each curve is a benchmark set that corresponds to 1 dataset and workload pair. Using the previous example, the CFD benchmark, using the 2M+ element model and 1000 iterations, forms one set of benchmarks. The ASPP will generate all the combinations for all the workloads and datasets using the classifications. Referring to the snippet of the configuration file seen in Figure 6.3 it can be assumed that since there are 3 models and 2 workloads the ASPP will generate 6 benchmark sets. Assuming a small cluster of 4 nodes (up to 16 cores) and the three classifications given in the snippets, ASPP will generate a minimum of 3 benchmarks per benchmark set. In total ASPP will generate 18 benchmark jobs to create a basic profile of the cluster.

These multiple curves can be used to predict run times for resource configurations not actually benchmarked. Multiple curves can be further expressed as 2 dimensional planes in a 3 dimensional space. The third dimension could be either dataset size or workload size. This form of benchmarking then provides the administrator or the users with a mechanism to predict run times at different resource allocations, even if their actual model does not fit the benchmarks that were previously run by ASPP.

Once the ASPP generates the job files for each job run, it begins to submit them

to the underlying batch scheduler. ASPP captures the returned job numbers and stores them in a file. This file is used as a flag to indicate to ASPP that the pre-processing has been done. ASPP then terminates and informs the administrator to re-run the benchmark once all the running jobs are completed.

6.2.3 Information Retrieval and Postprocessing

When ASPP is started with the configuration file described in section 6.2.1 it uses the configuration file-name as the key name for the whole benchmark series. The flag file shares this name, for example in case of a cfd.conf file, the flag file would be called .cfd.0. If the flag file exists, the ASPP launches the post-processor instead of the pre-processor, which is the default behaviour. For post-processing the ASPP should be run as a privileged user.

The post processor ensures that all the jobs defined within the flag file have completed and terminated naturally. In the event that the job state indicates an exit condition other than 0, the ASPP informs the administrator. It is left to the system administrator to ensure that the profiling jobs run through normally for the purpose of benchmarking. Some programs may output internal errors and still terminate in such a way that the batch scheduler does not identify an error. In the event when a job has ended due to an error, all the job files are available along with instructions on how to re-run them. Once an administrator has re-run the jobs, there is a script available to update the flag file with updated job numbers. This should be run before launching the post-processor.

There are conditions under which a job terminates with an error because the underlying system itself cannot cope with the load. A job could fail because the memory requirements for a particular resource combination is insufficient. These situations are handled by setting a flag to notify the post processor that such jobs can not be handled by this system. Furthermore some datasets or

workloads may not finish in an ”appropriate” time frame. There maybe no errors but a maximum wall clock time may be reached. This value is defined in the system configuration file.

Using the job numbers and meta data stored in the flag file, the post processor within the ASPP begins to generate records for each benchmark. ASPP processes the accounting logs for the job scheduler and collects the *wall-time* or running time for each of the jobs that was generated. Each record therefore has an Applications Name, Dataset size, Workload size, Resource Allocation, and finally a corresponding Running Time. If the maximum job duration for the system has been exceeded or the memory requirements are exceeded for a particular benchmark job then the profiler assumes the system can not handle such jobs.

These records are pushed to a MySQL backend and the data can then be accessed by any other application. The post processor can generate a report of execution information using an R (r-file) or Matlab (m-file) which details the performance and provides visual representation of the performance curves and surfaces.

6.3 Testing and Results

6.3.1 Test Platform

To evaluate the Application and System Performance Profiler testing of the toolkit was carried out on a small system (nodes partitioned out of the Eridani cluster) using two applications that are typical of the University of Huddersfield research environment. The underlying HPC system is an Intel®cluster with a head-node, 4 compute nodes and a gigabit backplane. Each node has an

Intel Q8200, 4 Core, 2.33Ghz processor with 8GB of 800Mhz RAM. The head node or control node of this cluster is separate from the 4 nodes outlined above. Each node runs CentOS 5 (final) that is installed locally and is managed using the Torque batch queuing system. Users' home folders and applications are delivered to the nodes using a network Gluster File system. The cluster management, data and MPI network paths all share the single gigabit back plane. A maximum duration of 14 days (1209600 seconds) has been set as the appropriate maximum run time for a job.

ASPP is used to benchmark this system across a range of allocated resources. It was decided to create seven bands or classifications. The user group supplied data sets were fitted to these classifications. The range in the bands are best guesses based on submission trends observed on our other HPC systems. The seven classifications are shown in Table 6.1.

Classification	**Range of Cores**
VVSMALL	1,2,4
VSMALL	2,4,6
SMALL	4,8,12,16
MEDIUM	4,8,12,16
LARGE	4,8,12,16
VLARGE	8,12,16
VVLARGE	12,14,16

TABLE 6.1: Classifications with corresponding Resource Ranges

ANSYS Fluent, version 13.0.0, was utilised as the first application. ANSYS Fluent is a proprietary computational fluid dynamics software and is characteristic of the computational fluid dynamics (CFD) application executed on the Queensgate Grid (QGG), as discussed in Section 5.2.2. The dataset provided by the end-users involves the calculation of forces on an object caused by airflow in a rectangular tube. The coarseness of the model has been varied across the typical sorts of mesh our end-users simulate. To this end there are 7 data sets that are shown in Table 6.2.

Classification	Elements	Shorthand
VVSMALL	312500	0.25M
VSMALL	496190	0.5M
SMALL	1014429	1M
MEDIUM	2193408	2M
LARGE	4093656	4M
VLARGE	8109956	8M
VVLARGE	16164981	16M

TABLE 6.2: ANSYS Fluent Benchmark Inputs and Classifications

Classification	Atoms	Shorthand
VSMALL	500	0.5k
MEDIUM	4000	4k
VLARGE	32000	32k

TABLE 6.3: DL_POLY Benchmark Inputs and Classifications

The second application used is DL_POLY Classic (Smith & Todorov, 2006). DL_POLY is a general purpose molecular dynamics (MD) simulation software developed by Dr I. T. Todorov of the Science and Technology Facilities Council (STFC). The model provided by the Applied Sciences user group are simulations of bulk MgO. There is a single workload, which has 1000000 time steps, where each step is 0.001 pico-seconds. Three datasets have been provided. In each dataset the number of atoms are varied. According to the chemistry users guidance these models have been given three classifications from Table 6.1. The DL_POLY models and the associated classifications can be seen in Table 6.3.

These benchmarks and corresponding classifications fit within the narrow window of the typical models and workloads generated by the CFD and MD user base at the University of Huddersfield. These values and ranges may differ at other institutions (Holmes & Kureshi, 2010).

Classification	Dataset	Cores	Duration
VVSMALL	0.25M	1	1956
VVSMALL	0.25M	2	1254
VVSMALL	0.25M	4	952
VSMALL	0.5M	2	2154
VSMALL	0.5M	4	1499
VSMALL	0.5M	6	1011
SMALL	1M	4	3032
SMALL	1M	8	2800
SMALL	1M	12	1030
SMALL	1M	16	834
MEDIUM	2M	4	6636
MEDIUM	2M	8	3390
MEDIUM	2M	12	2220
MEDIUM	2M	16	2889
LARGE	4M	4	12573
LARGE	4M	8	10673
LARGE	4M	12	4095
LARGE	4M	16	5638
VLARGE	8M	8	12975
VLARGE	8M	12	8142
VLARGE	8M	16	11292
VVLARGE	16M	12	17024
VVLARGE	16M	14	15858
VVLARGE	16M	16	12873

TABLE 6.4: ANSYS Fluent Benchmark Results by Classification for Single Workload

6.3.2 Profiling the CFD Application

The various datasets outlined in the previous section were run via the ASPP toolkit. In the interest of space and clarity one workload of a 1000 iterations has been explained below. The system generated a total of 24 benchmarks against this workload size, three benchmarks each for the VVSMALL, VSMALL, LARGE and VVLARGE classifications and four benchmarks each for the SMALL MEDIUM and LARGE classifications. The increased granularity for the latter classifications is due to the wider band of resources provided. The tabulated results for the benchmarks can be found in Table 6.4.

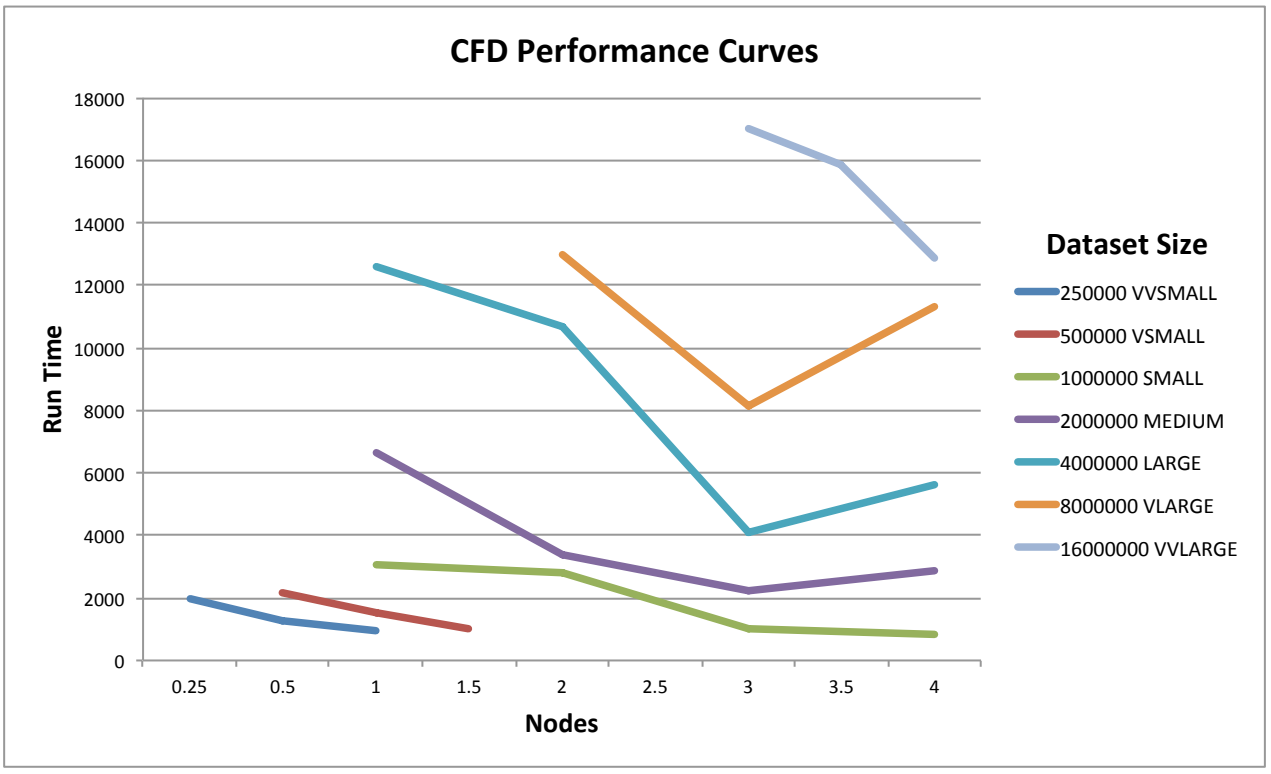

FIGURE 6.4: CFD Performance Curves as provided by ASPP m-file

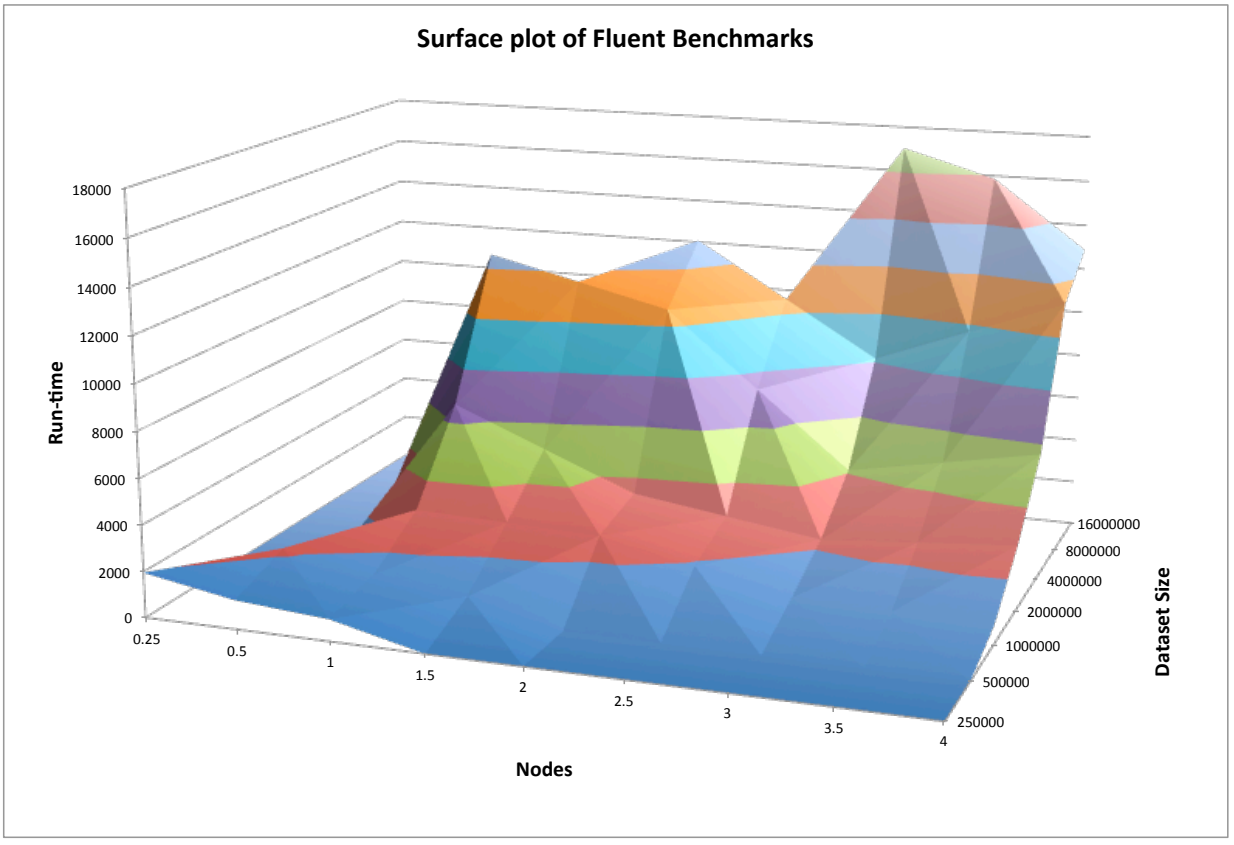

FIGURE 6.5: Surface plot for CFD Performance as provided by ASPP m-file

Classification	Dataset	Cores	Duration
VSMALL	0.5k	2	29102
VSMALL	0.5k	4	16570
VSMALL	0.5k	6	45815
MEDIUM	4k	4	657791
MEDIUM	4k	8	142997
MEDIUM	4k	12	100019
MEDIUM	4k	16	80628
VLARGE	32k	8	1209601
VLARGE	32k	12	1209601
VLARGE	32k	16	1209601

TABLE 6.5: DL_POLY Classic Benchmark Results by Classification for Single Workload

6.3.3 Profiling the MD Applications

Based on feedback from the DL_POLY user base, the ASPP was directed to generate 3 batches of benchmarks. These were classified as VSMALL with 500 atoms, MEDIUM with 4000 atoms and VLARGE with 32000 atoms. As defined in the ASPP configuration VSMALL classifications get mapped to 3 possible hardware combinations; a MEDIUM classification gets mapped to 4 hardware combinations; and VLARGE classified benchmark inputs get mapped to 3 hardware configurations. To benchmark DL_POLY Classic across our testbed system ASPP generated 10 benchmark jobs. These jobs and their results can be found in table 6.5.

6.3.4 Discussion

In this section we evaluate the results of the ASPP system and the system itself as a stand-alone software.

The results of the Fluent benchmarks can be visualised as stacked curves, shown in Figure 6.4. As can be seen by Figure 6.4 the difference between

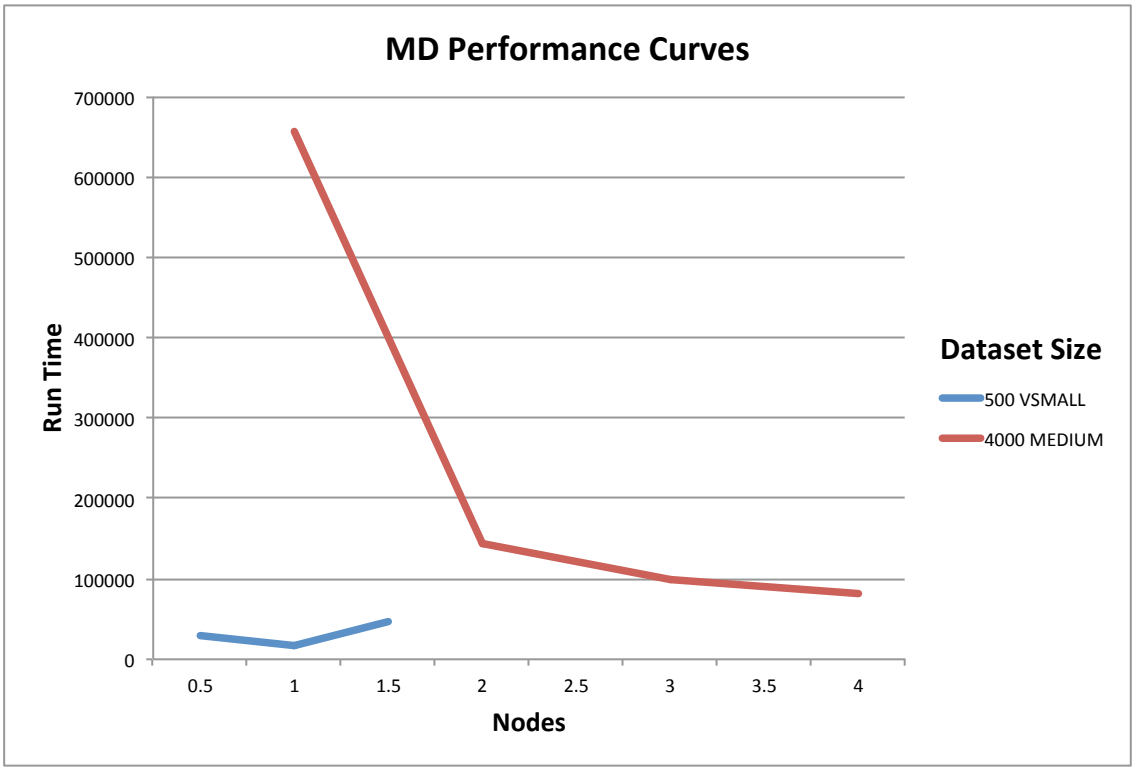

FIGURE 6.6: MD Performance Curves as provided by the ASPP m-file

4 cores and 12 cores for a SMALL dataset is insignificant and the user may decide to submit multiple jobs requesting 4 cores to maximise his throughput, rather than trying to increase the speed of each job but having to wait longer. In the case of the MEDIUM dataset both the 4 and 12 Core configurations may not be preferable as the former takes too long per simulation and the later wastes resources. In the case of MEDIUM datasets, eights cores appears to be a good compromise. This information once provided to the user base becomes invaluable, as it leads to better utilisation of the researchers time and the system itself. It can also be seen that for different dataset sizes the performance/scalability characteristic of the application changes. This proves our assertion that algorithmic models for performance profiling do not give adequate results and that real application and dataset profiling gives a more accurate measure of performance. The performance curve for a 2M element benchmark has the opposite shape as the 4M element benchmark.

Using different sized datasets provided by our mechanical engineers we classified them as VVLARGE, VLARGE, LARGE, MEDIUM, SMALL, VSMALL and VVSMALL. Using ASPP on our 4 node test cluster we profiled each classification of dataset across different combinations of cores. The ASPP demonstrated that there is good scale up on our test system up to 3 nodes. The only exception to that is for the VVLARGE family of simulations, which can scale to 4 nodes.

A 2D representation of the results of the DL_POLY can be seen in in Figure 6.6. As the wall clock time of the VLARGE benchmarks exceeded the 14 days upper limit the ASPP assumes that for this size dataset and workload the system can not reach a result. As Figure 6.6 shows for VSMALL models around the 500 atoms mark it is more expensive to scale beyond 4 cores on the test system. Whereas for the MEDIUM sized dataset with 4000 atoms it is always beneficial to scale up. However beyond 8 cores the speed up is not significant.

The models provided by our applied scientists fit their classifications of VSMALL, MEDIUM and VLARGE. The graph in Figure 6.6, generated by the ASPP output m-file, shows that while the medium sized simulations benefit from scalability, our test cluster is inefficient for small problems and is unable to cope with significantly larger problems.

The ASPP generated figures can help system administrators and the research community to make better use of the system. Users can be clearly advised of the scalability of the system. The curves also provide the users with estimated runtimes for their datasets. Users can decide if they would rather submit a job consuming more cores or submit more jobs using fewer cores, based on the estimated runtimes that have been derived by the benchmark.

Furthermore the results of ASPP can be used to identify different requirements for various applications, which could leads to more informed purchasing decisions. When purchasing a new system the end-users can evaluate the new

system performance and compare it with past performances. By not relying on just the peak performance of the system (kernel or application specific) end-users can also predict the throughput of the system. In addition, using the benchmarks above, end-users can make license purchasing decision. As best performances are observed at a certain number of cores, licenses can be purchased as multiples of the optimum core count, leading to increased efficiency of license utilisation.

During the development of the Mouldable Scheduler the figures generated by the Application and System Performance Profiler lead to a greater insight into the QGG user base's requirement and led to significant changes in the way the internal high end compute systems were utilised. It is hoped that with these performance profiles and estimated run times, HPC usage on traditional clusters can be made greener, job schedulers can be made more efficient and migrations to cloud environments can be appropriately preplanned as running costs can be estimated.

6.4 Summary

In this Chapter the efforts undertaken to create a mechanism to generate basic performance profiles for applications running on Research Computing Systems (RCS) has been outlined. To achieve the goal of autonomic allocation of resources to jobs and thus creating a job scheduler capable of mouldable scheduling this is the critical first step. Using datasets and workloads provided by user communities within the University of Huddersfield new insight was derived regarding the performance of the local application base. So in addition to providing key data to the mouldable scheduler (described in Chapter 8) the ASPP can be used as a stand alone system.

These benchmarks provide anchor points for an applications performance characteristics. With the performance data stored in MySQL database, plugins can be created for the job scheduler to access this vital information. For dataset sizes not benchmarked by the Profiler the mouldable scheduler will need to interpolate and extrapolate. It is too expensive to benchmark the entire system but creating this baseline of resources required and expected run times is enough to achieve the goals set out in this thesis. By providing more accurate data to the scheduler a system administrator can expect better throughput and utilisation within their system while maintaining a high quality of service.

In the next chapter the development of the scheduling algorithms is discussed as well as the testing platform that is implemented for simulation and testing purposes.

Chapter 7

Workload Manager Simulator

7.1 Introduction

In order to test the effectiveness of the various scheduling algorithms and approaches a flexible HPC Workload Simulator is required. In this section the design, development and testing of the toolkit Cluster Discrete Event Simulator (CDES) - an open framework workload simulator is discussed.

To meet the various needs and limitations outlined in Section 3.5 the CDES system was developed as a strong candidate for HPC workload simulation. This chapter is based on work that has been done in collaboration with a Grid Scheduling project and published as a paper at the DS-RT 2014 conference (J. Brennan, Kureshi, & Holmes, 2014). Built around an open framework, CDES can take system definitions, multi-platform real usage logs and can be interfaced with any scheduling algorithm through the use of an API. CDES has been tested against 3 years of usage logs from a production level HPC system.

Section 7.2 covers the design aspects of the CDES. The implementation methods are included in Section 7.3. Testing and Validation of the simulator are presented in Sections 7.4 and 7.5 respectively.

7.2 Design

As discussed in Section 3.5 market available workload simulators are unable to use real logs to evaluate new scheduling algorithms or paradigms. To meet this requirements the Cluster Discrete Event Simulator was envisaged. The main objectives in designing the CDES was to develop a system which would be able to:

1. Create a simulated system of any required configuration.
2. Read in job information from historic Torque accounting logs.
3. Schedule the jobs to an appropriate simulated resource.
4. Allow for different scheduling algorithms to be interfaced.
5. Remove 'completed' jobs from the simulated resource.
6. Produce updated modified logs for simulated system.
7. Match features and performance of existing schedulers

To ensure a truly flexible product extra care was taken not to hard-code any aspect of the system so that an end user can customise the system to their requirements. For example, a simulated system would be constructed of a number of nodes with a variable number of cores. However, if the system was hetrogeneous then all nodes might not have the same number of cores. Hence, the

simulator needed to be capable of simulating a system that was heterogeneous and in some cases distributed in nature.

Essentially CDES is designed to take a real user workload and apply a user specified scheduling algorithm to a user defined system layouts and configurations - while mainting job execution times. What CDES is not designed to do is predict changes in a single jobs run time due to changes in interconnect, processors and memory. If a user needs to make the system adapt to such changes then those need to be written into the scheduling algorithm.

7.3 Implementation

CDES has been implemented to read the historical logs pertaining to the completion of jobs. These records contain most of the job information, including times and requested/consumed resources. The logs are read into a dynamic table of tables (hereby known as the initial table). In order to maintain the data integrity of the user behaviour, the time at which a job was created on the system, referred to as ctime within Torque, is never modified. The simulator, as its output, needs to produce new logs with modified records for the start, and end of a job. This output is generated based on the new simulated start time for a job and utilises the historic duration of the job to calculate the end time.

The final logs produced by the CDES include all pertinent information gathered from the historical logs such as UID, job ID, time job was created etc. The simulated logs also provide, where appropriate, modified job completion times. All times within these logs adhere to Unix epoch standard. For initial functionality (and to accurately match the existing HPC systems within the University of Huddersfield) the default scheduling algorithms within the CDES is the First Come First Served (FCFS) and FCFS with Backfilling.

Table Name	Properties
Initial Table	Holds the raw job information as read in from real job logs
Queued Queue	Holds the full job information as taken from the Initial Table. This job has already triggered a loop of the simulator. No values are changed
Sorting Table	This queue includes any job that has gone into either a running state or a queued state. The queue holds the job number, a flag for running or queued and a trigger time for the simulator to decide exection time (Queued job trigger = submit time, Running job trigger = expected end)

TABLE 7.1: Details of Tables/Queues within the CDES system

Along with the *initial table*, CDES maintins several other tables that simulate queues. These are described in Table 7.1.

The approach applied to the simulator system has been to activate the system logic upon the arrival of a new job. As the simulator iterates the table of unprocessed jobs every *new job* acts as a trigger to the system. The system begins to work its way down the initial table (simulating the arrival of a new job). The first job will trigger the system and as there will be nothing in the master queue this new job will enter into the system. Before creation, job requirements will be matched against the system and the best fitting nodes (rows in the system table) will be allocated. The system will evaluate the completion time and in the sorting table will put the job number and the completion time and a flag to depict a running job. The sorting table is kept sorted on its trigger time.

When no more jobs exist then the system clears up the queues by flushing out jobs. This has been the most effective and least CPU intensive mechanism to model large usage data. A second approach that was attempted, during early design implementations, was to have a counter increment through the epoch time and trigger system behaviour appropriately. This creates operational overheads and is susceptible to incomplete runs.

Using the trigger approach (as seen in Figure 7.1, usage logs from the physical HPC system are pushed into an array (initial table) and sorted with respect to creation time. The system has three further arrays or queues, the running jobs array, the queued jobs array and a master array outlining job positions and trigger (when action needs to be taken) times. At start up CDES also looks at the system definition file and generates a 2D array which mimics the system (e.g. 2 nodes with 4 cores, and 2 nodes with 8 cores will result in a 4 row array with 2 columns of length 4 and 8, respectively, being generated).

In the next cycle when a new jobs arrives the system compares the ctime (arrival time) for the new job with the trigger time at the top of the master list. If the trigger time is in the past, when compared to the new job the existing job is popped from the array of running jobs and the master table (depicting an eviction from the system). Based on the type of scheduling algorithm that is in place the new job is either executed or the queued jobs (if they exist) will be dealt with. If there are no resources available then this new job will be pushed into the queued queue. It will be added into the master queue as well. It will have a queued flag and its trigger time will be the creation time. This way the master queue can be kept sorted for running jobs and queued jobs with increasing trigger times.

Assuming an FCFS system (as depicted in 7.1) and some jobs in the running and queued queues, on a new job trigger:

1. any running job with an earlier end time to the trigger to be popped and all queues resorted. Its logs will be written out to a new file.;
2. if there is still a running job at the top of the master list 1) is repeated.
3. the top queued job to be pushed into the running queue if space is available. Its new completion time calculated based on what time a running job was popped, making space for the queued one;

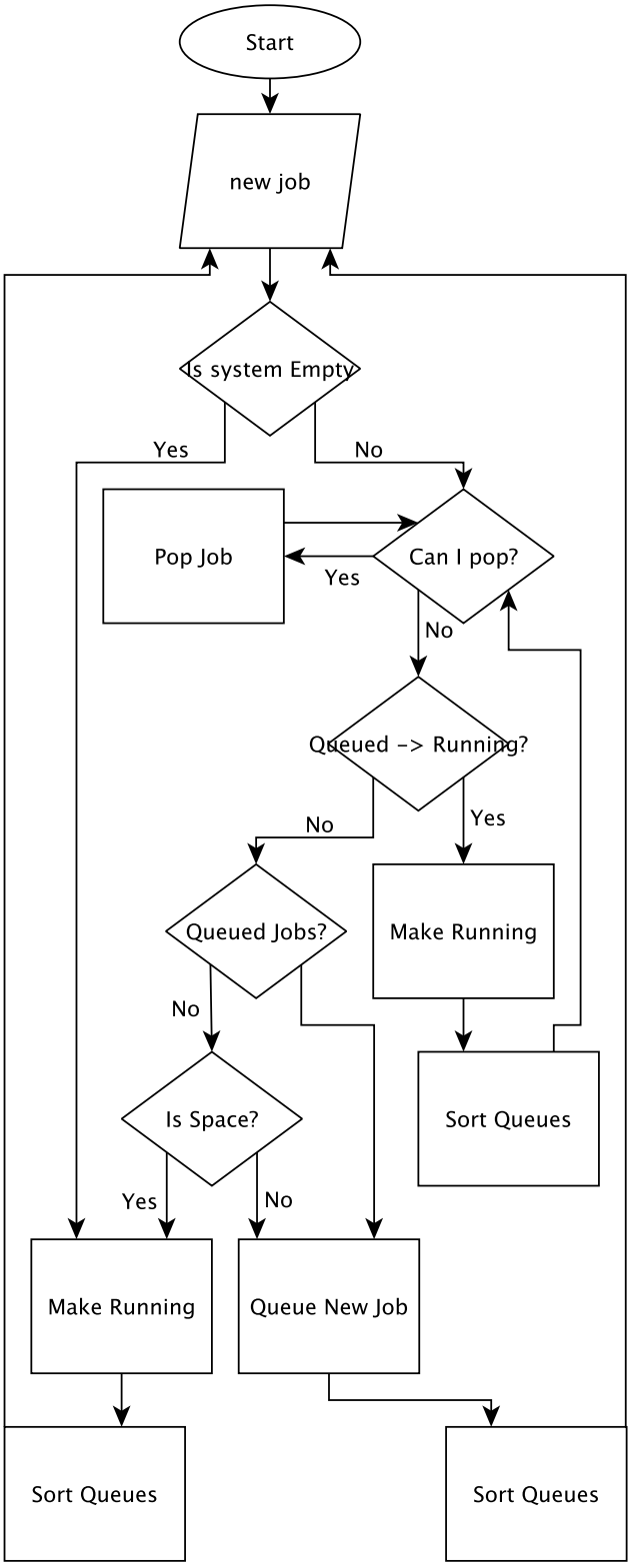

FIGURE 7.1: Flowchart showing system behaviour when interfaced with an FCFS algorithm

4. any subsequent ”newly” calculated trigger times to be evaluated as depicted in 1).

5. with no more pops or pushes the new job would either be queued (if there are still jobs waiting to run, or not enough resources were available) or set to run (if there are no more queued jobs and resources are available).

With no more triggers the system would go through steps 1-5 as outlined above popping running jobs, pushing queued jobs and finally popping those jobs, creating appropriately updated accounting logs.

7.4 Testing

The CDES system has been tested against the entire High Performance Computing (HPC) workload on the Eridani system (3 years worth of job logs). Once again, for clarity, only results from jobs completed within 2013 are preseneted here to establish the performance of the CDES system.

Arrival rates of jobs will always remain the same, because the CDES can not modify the time a job had been historically created. This was taken as the first validation of correct behaviour. As highlighted in Figure 7.2 this comparison was encouraging. Visibly there is also a very close correlation between the amount of jobs submitted by users and the subsequent completion of those tasks.

Considering job completion data for 2013, Figure 7.3 shows the results, on a logarithmic scale, for the original Torque logs and those for the simulated completion, they are almost identical showing only minor discrepancies in the number of jobs completed within each interval. These discrepancies are explained further down as the completion rates are broken down into more manageable sections. The following are four sections analysed here:

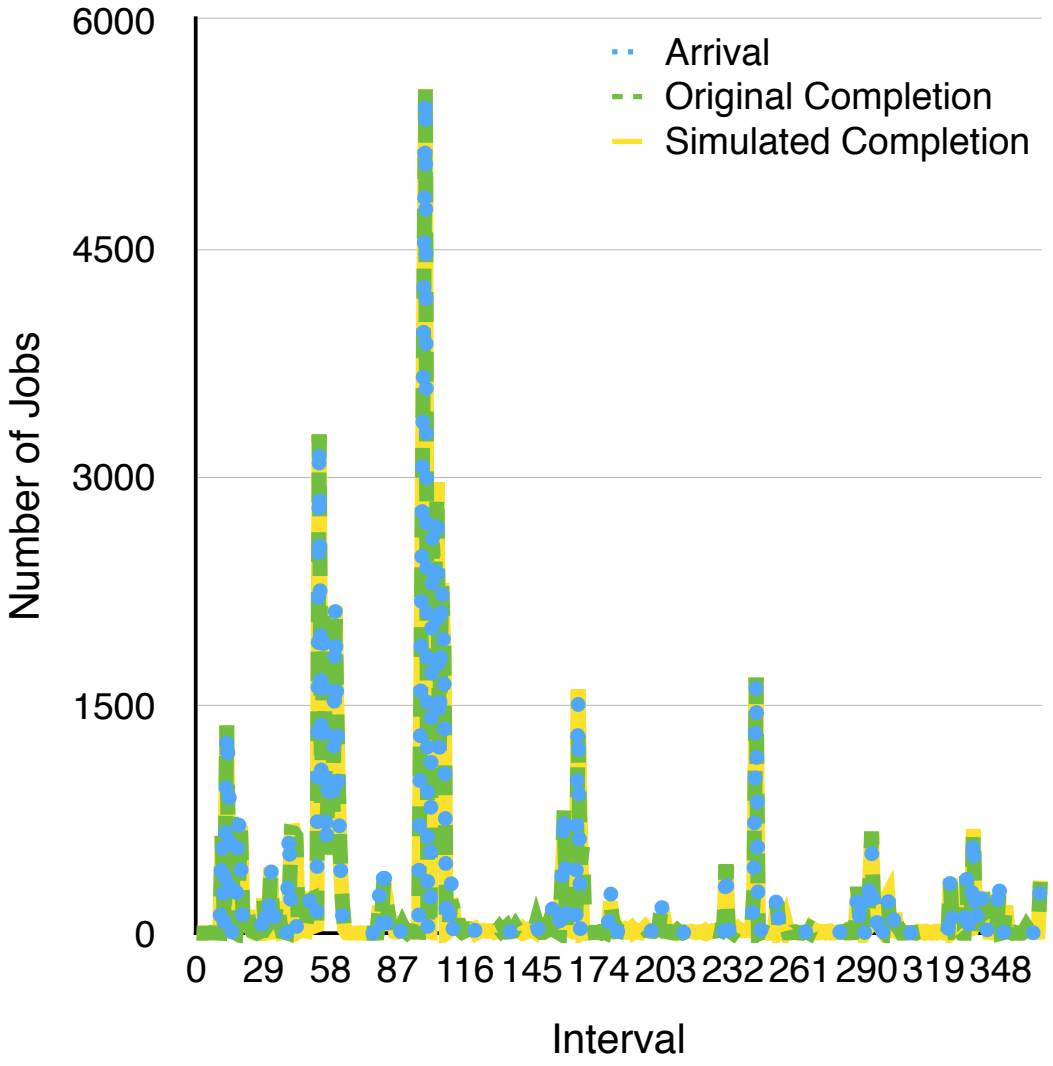

FIGURE 7.2: Comparison of Arrival and Completion Rates over 2013

1. 2013 or Annual Period

2. Average Load (AVG) Period

3. User Intervention (UI) Period

4. System Intervention (SI) Period

7.5 Validation

In the first period of focus, this is a period where the system is considered under an average load (AVG), which covers the days between 11/11/2013 to 08/12/2013, shown in figure 7.4. This period is considered average load because the systems utilisation is around the 70% mark. This corresponds to the systems average utilisation over the year. This period also has a mix of parallel, core-serial and node-serial jobs from users in different disciplines. During this AVG period the simulated results exactly match those of the real system, as depicted by the perfect overlap of the lines. This result is extremely encouraging.

A two week period between March and April 2013 shown in Figure 7.5 this is another period with no system or user intervention in the execution of jobs. There is a 100% correlation of job completion rates, as depicted by the matching blue and green peaks. This is also the time period outlined in Section 5.4 as the period where the typical users and typical applications all run. These tests have shown that with simple FCFS with backfilling CDES is able to accurately predict the performance of the HPC system.

Figure 7.6 shows a period that has been designated as the System Intervention (SI) Period. During this period (in late April) the workload was exceedingly heavy and the Maui scheduler resorted to out-of-order execution due to Fair Use policies. In this one month period 5564 jobs had been completed on the Eridani system within a single day and a total of 24,716 jobs completed across the period. Within this period the fair share component of Maui had an impact upon the real system, which created discrepancies between the two datasets. As the granularity of the samples is decreased these discrepancies also decrease. Since the FCFS algorithms implemented in CDES did not have a fair share feature, it could not make decisions within that scope and hence the simulation deviated from the real results. The impact of this manifested in a total difference

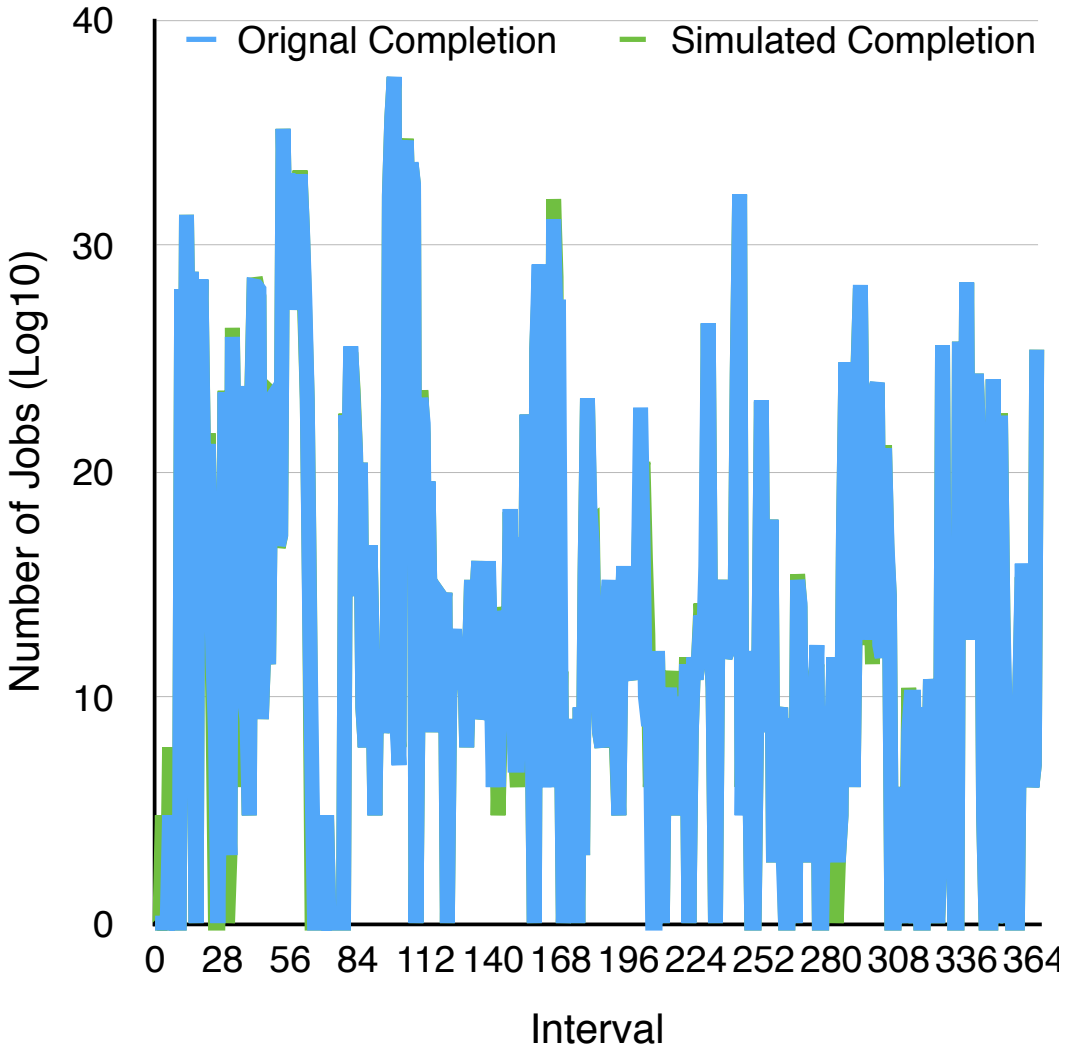

FIGURE 7.3: Comparison of Original and Simulated Data for 2013 (Log Scale)

in completion over the period of 286 jobs which was only 1.16% of the period total.

The period in 2013 that created the largest differences in the results, referred to as the User Intervention (UI) period, was between May and August, shown in figure 7.7. During the UI period there was a difference of 1816 out of a total of 5861 jobs. This figure approximates to around 31% which is quite unfavourable. However taken together with the raw data this problem can be explained. During this time a large number of the jobs submitted to the real system were subsequently deleted by the user. As this behaviour was not accounted for during the

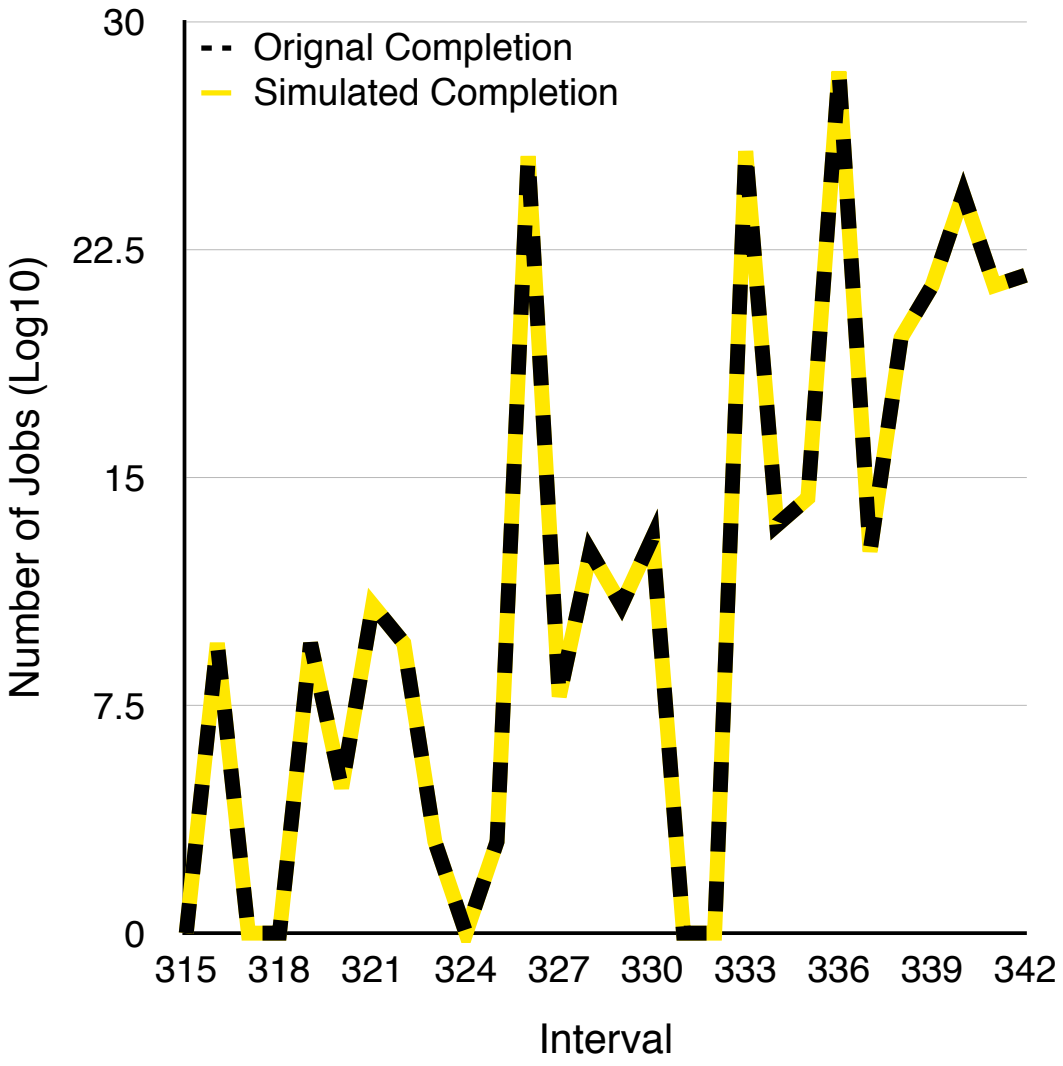

FIGURE 7.4: Comparison of Original and Simulated Data during period of average system load (AVG) [Log Scale]

development of the CDES the software could not respond to those events, this will be addressed in further revisions of the simulator.

The discrepancies within the defined periods only actually represents a 4.89% variance in completion rates within the results gathered for 2013, this includes the large 31% difference observed during the UI period. A total of 62376 jobs were completed within 2013, and in the 24hr interval results the average difference between real and simulated data was only 8.4 jobs per day. This gives an average annual difference of 3052 jobs which is $<$5% of the total number of jobs processed. These differences were entirely due to the CDES lacking

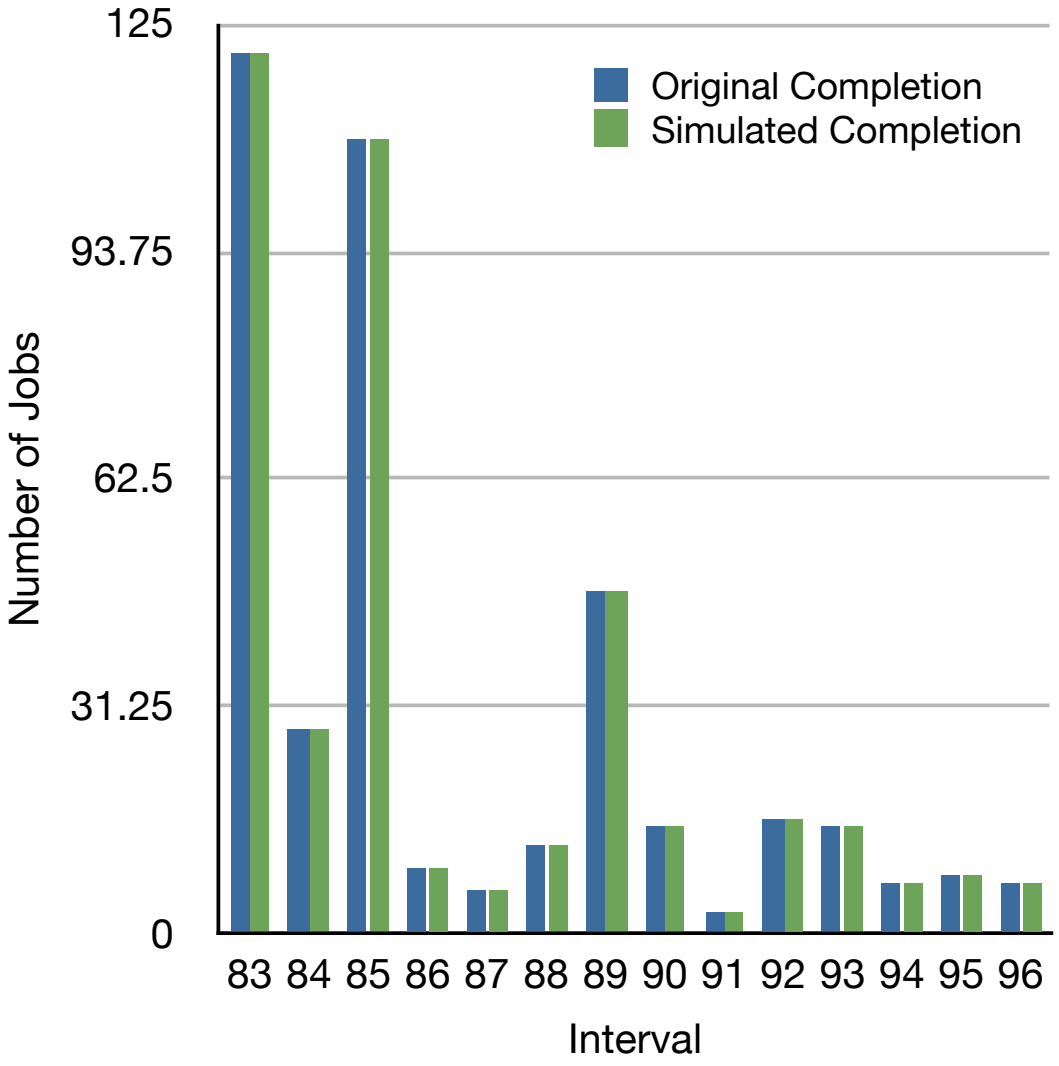

FIGURE 7.5: Comparison of Original and Simulated Data: 2 Week Period 23/03/2013 - 06/04/2013

implementation of a fair share component and handling of job deletions.

Tables 7.2, 7.3 and 7.4 include aggregated data from the periods included in the previous graphs. The tables show the number of jobs completed within a given period; the number of intervals within that period; the number of variant jobs, those jobs which the CDES did not complete in the same interval as the real system; the average number of jobs completed per interval (Jobs/Int) and the average variant jobs per interval (VJ/Int).

Table 7.2 shows that when using 15 minute intervals the average number of

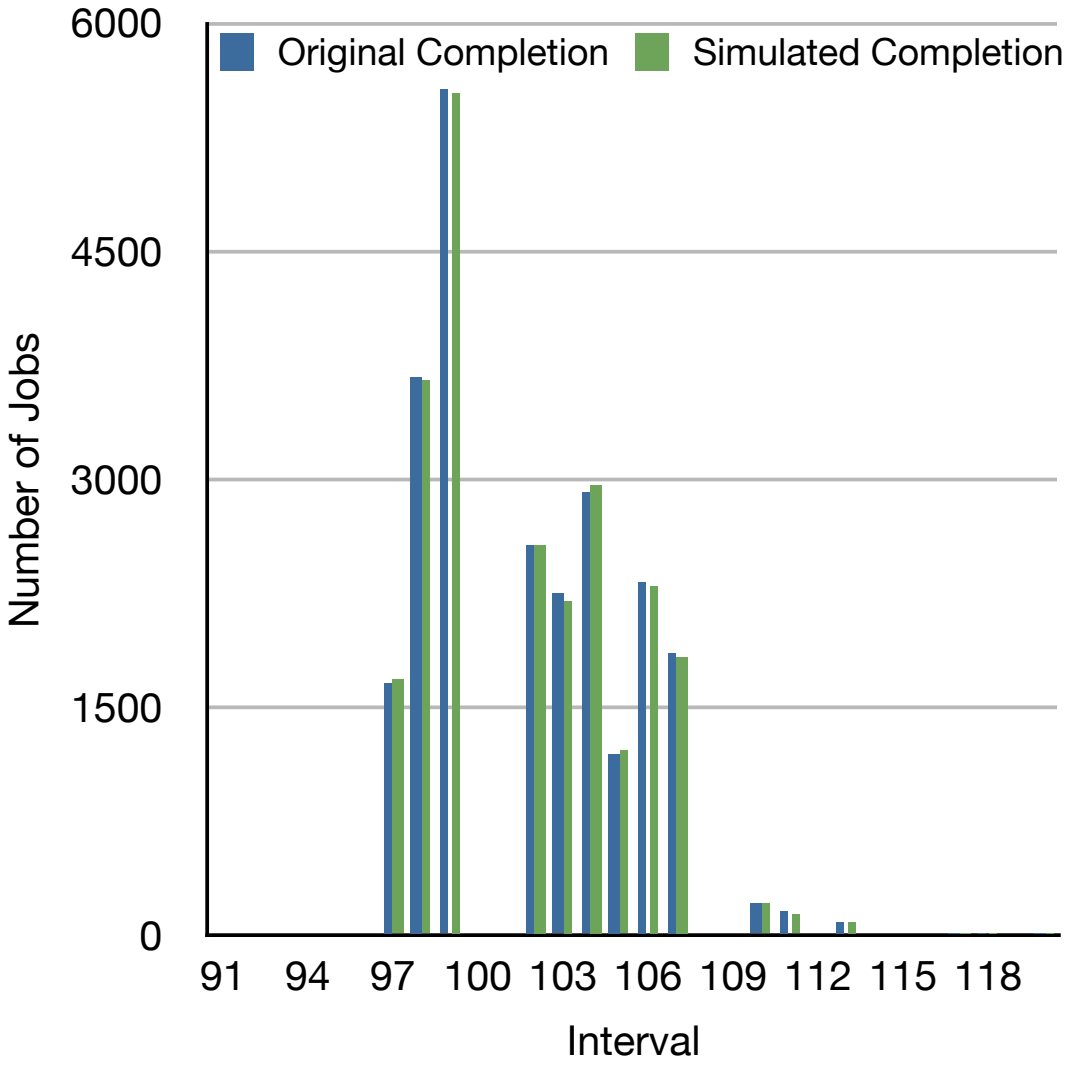

FIGURE 7.6: Comparison of Original and Simulated Data for period with System Intervention (SI)

variant jobs with the actual number of average jobs was not encouraging, particularly within the UI period. Such a short sample period is however very fine grained and not very representative of an overall system view. The results improve significantly when looking at 1 hour samples, in table 7.3, showing an 11.2% improvement in overall correlation. The final table, examining 24 hour intervals, shows that during the UI period the results still differ significantly. However all the other periods show a very strong correlation with the result from the real system, particularly within the AVG period where there is no deviation at all.

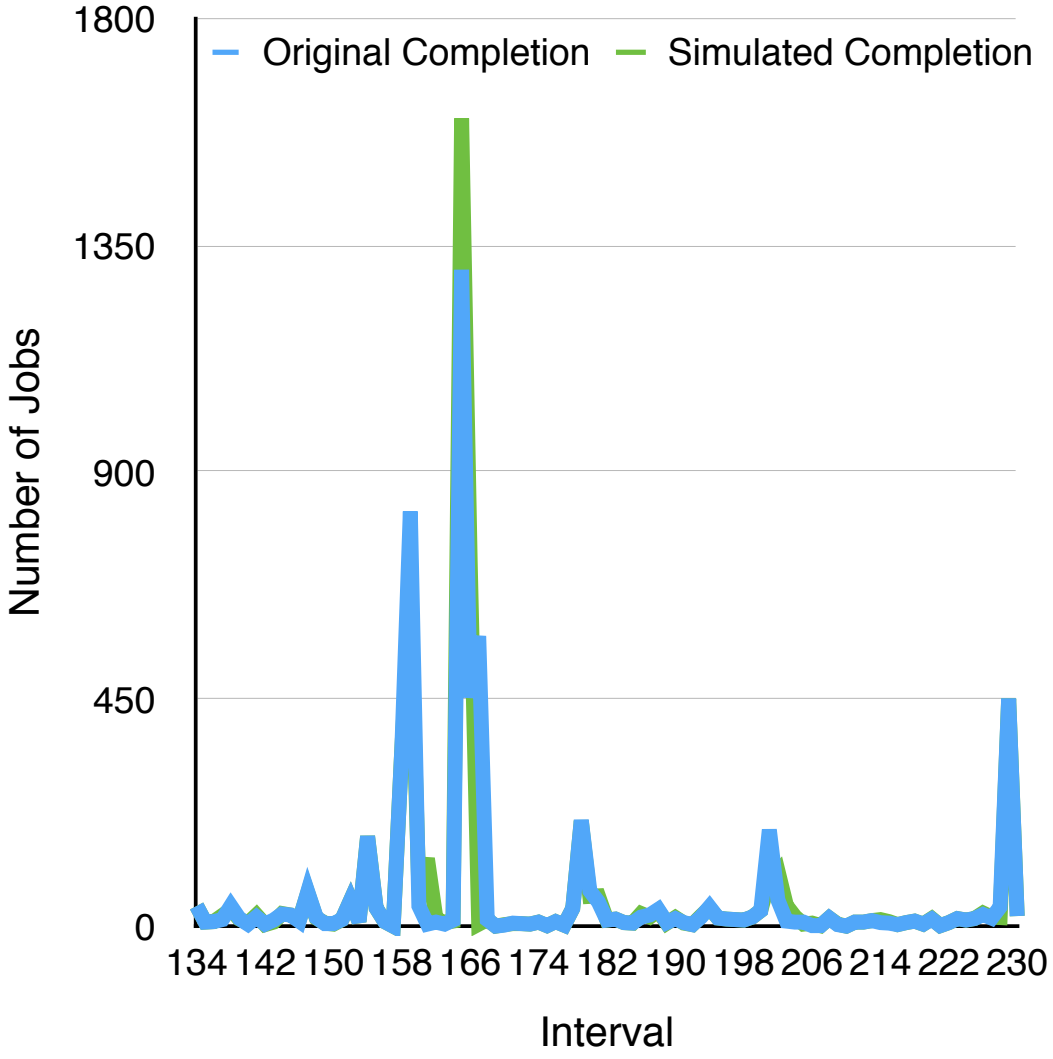

FIGURE 7.7: Comparison of Original and Simulated Data: for period with heavy User Intervention (UI)

These results allow confidence that the simulator would be capable of predicting future system behaviours, for a variety of hardware configurations.

Period	**# of Jobs**	**# of Intervals**	**Variant Job**	**Jobs/Int**	**VJ/Int**
2013	62373	35054	16148	1.78	0.46
AVG Period	2363	2592	560	0.91	0.21
UI Period	5844	9216	5370	0.63	0.58
SI Period	24699	2784	2814	8.87	1.01

TABLE 7.2: 15min Interval Snaphots

Period	# of Jobs	# of Intervals	Variant Job	Jobs/Int	VJ/Int
2013	62373	8766	10742	7.12	1.23
AVG Period	2363	648	140	3.65	0.22
UI Period	5844	2304	4572	2.54	1.98
SI Period	24699	696	1546	35.49	2.22

TABLE 7.3: 1hr Interval Snaphots

Period	# of Jobs	# of Intervals	Variant Job	Jobs/Int	VJ/Int
2013	62373	365	3052	170.88	8.36
AVG Period	2363	27	0	87.52	0
UI Period	5844	96	1816	60.88	18.92
SI Period	24699	29	286	851.69	9.86

TABLE 7.4: 24hr Interval Snaphots

7.6 Summary

The Cluster Discrete Event Simulator provides a platform to test the mouldable scheduler against a realistic workload. It has been designed in such a way that researchers and system administrators can utilise it to model actual HPC workloads. The software allows for users to plug in their own designed scheduling algorithms for testing (as it currently support FCFS, FCFS with aggressive back filling and the mouldable scheduler. Any sort of workload can also be supplied to the software to predict the performance of the system. As CDES very closely matches the performance of actual HPC systems it will form the basis for the analysis of the mouldable scheduler. The system has been presented at the DS-RT 2014 conference in Toulouse France. The datasets outlined in this Chapter have been anonymised and made publicly available so that the results can be further validated.

Chapter 8

Rule Based Mouldable Workload Manager

8.1 Introduction

High Performance Computing (HPC) systems have tremendously grown in size, and increasingly they are required to manage diverse workloads. The one given constant over the years has been system size (or available processing end points)[1]. With the advent of cloud computing this fixed parameter is no longer static, and system are going to have to adapt and make decisions to scale up.

As described in Section 1.1, HPC systems are rigid and can not cope with dynamic situations where a system is either elastic or is shared between different user groups with a fixed Quality of Service (QoS). The way systems maintains the QoS is by keeping at times very large portions of the systems idle. It does this because it is unable to guarantee or even estimate the expected run times of jobs on the system. This leads to frustration amongst some users who see

[1] The only exception to this is if a node is down but this can be seen as a constraint - similar to an indefinitely busy node

the system as idle but have their jobs queued. From a management perspective systems remaining idle is a waste of money.

Scheduling algorithms as described in Chapter 4 have evolved over the last few decades to improve system management. However inevitably these approaches all require user prescribed information to make scheduling decisions. Even within the mouldable computing paradigm authors have required the users to give a range of resource allocation possibilities and then the system chooses the best one. While a step in the right direction, this approach suffers from the same bad-put observed in static scheduling. Unless their simulations crash users are inherently not diligent enough to modify their submission files when they modify their data and application parameters. Further user parameters exist in the form of *'number of cores or memory'*, and *'wall time'*. These is not native information for early researchers. A researcher from chemical sciences is more likely to know *'atoms or molecules'* and *'time steps or iterations'*. The balance of getting performance vs. run-time for their dataset and their application against a set of resources is best described as *gut instinct* that a researcher develops over time. However as their research advances and becomes more complex, users migrate to newer and bigger systems making their previous intuition of no use in the new context.

A novel and significant approach to counter these sort of allocation problems is described Section 4.5 from the paper by Srinivasan et al. The use of the Downey model (Downey, 1997) to estimate the performance of an application based on algorithm parallelism help Srinivasan et al. to mould a synthesised trace log of real jobs and achieve a better turn-around-time (Srinivasan et al., 2003). There are several shortcomings and omissions in this approach. Primarily the Downey model is system agnostic. So it assumes ideal scalability and not actual. Actual scalability is usually constrained by network overheads, data location and data read/write speeds. Further the Downey model is based

around algorithm scalability and most software vendors would argue that they have included special optimisations to improve performance and accuracy, and therefore a comparison to a standard algorithm is unfair. Because their approach is based on Schedule time aggressive fair share their approach ignores the amount of time a job sits in the queue before its time to be considered. Finally in their work Srinivasan et al. have not described implementation methods of their algorithm. Their synthesised logs approach makes it difficult to visualise the actual implementation method as the parameters used to mould are not intuitive for a user.

In this Chapter a new method for Mouldable Scheduling, influenced by Srinivasan, is described. Using the real application benchmarks described in Chapter 6 and the workloads described in Section 5.4 the mouldable scheduler described below will aim to improve the performance of the turn-around-time of the system. Section 8.2 explains the design of the mouldable scheduler. There are two further subsections that include real world implementations to capture user input (8.2.1) and a detailed break down of the mathematics behind the moulding component (8.2.2). Finally in Section 8.3 the testing and results of the mouldable scheduler are described in detail.

8.2 System Design

The mouldable scheduler is based on discrete components and is laid out as seen in Figure 8.1. This system will request the application name, job family name (simulation name), dataset size and workload size from users at submit time. The system works on a two stage moulding mechanism. The system flowchart is shown in Figure 8.2.

At job submition the job scheduler will utilise the results of Application and System Performance Profiler (ASPP) when making its scheduling decision. As discussed in Chapter 6, the results of ASPP are stored in a database. This output is in the form of points on an application performance characteristic curve. Using the metadata provided in the job-submission file the scheduler can query the database and request the coordinate points for relevant benchmarks. Using these points (coordinates on a nodes-vs-time graph) the scheduler can model the curves. This request and generation of the curves is explained in detail in Section 8.2.2.

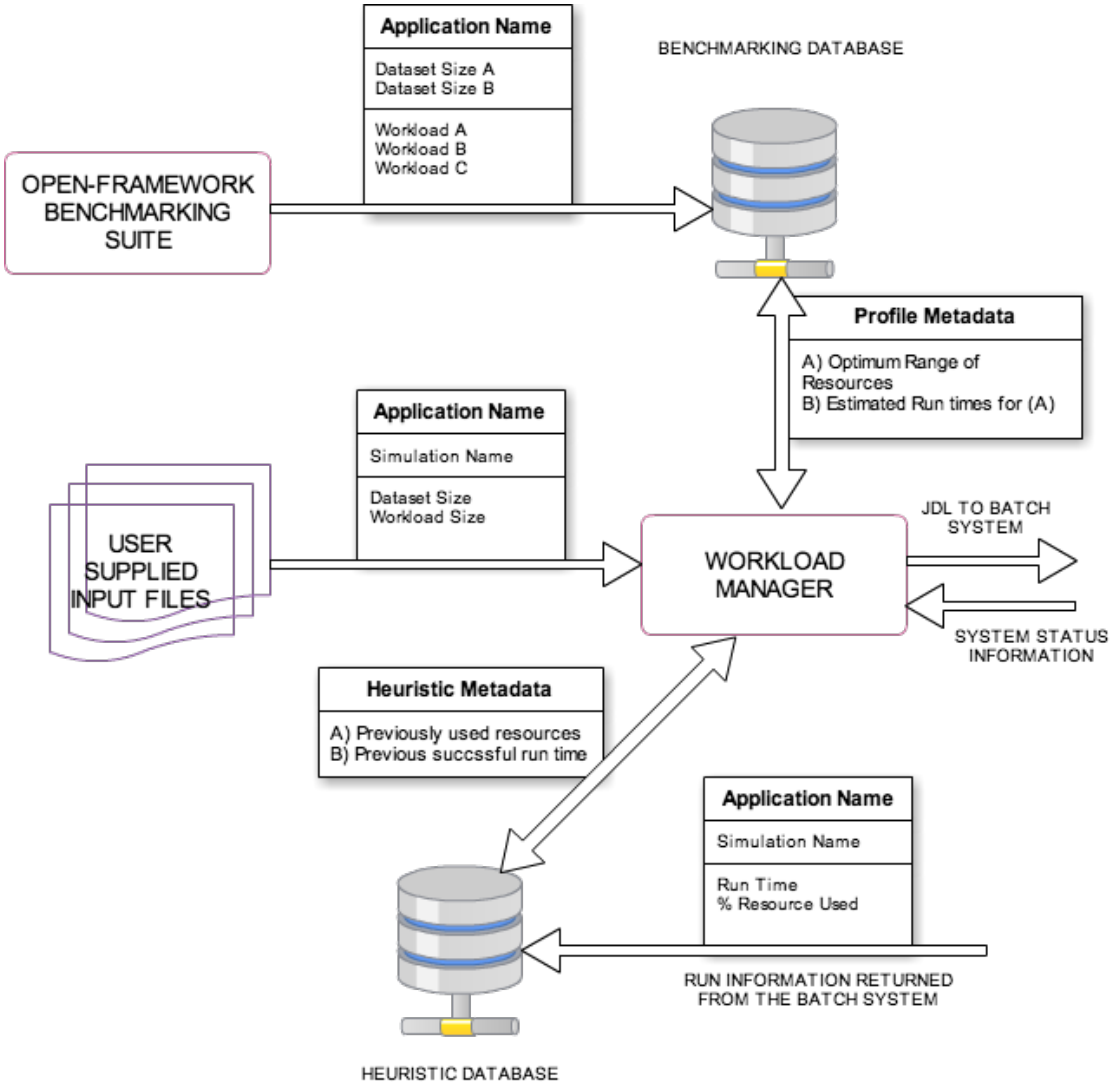

FIGURE 8.1: Mouldable Scheduler System Layout

Once the curves have been evaluated the workload manager can determine a range of optimum resources to allocate to the job. This range gives the scheduler the maximum allocatable resources with expected runtime, the minimum allocatable resources with runtime, and all run times for resource allocations in the middle. At this stage the scheduler becomes application agnostic. For the purposes of this research, the Mouldable scheduler will be tested using a First Come First Served (FCFS) algorithm and all example decisions will be explained using this premise.

The generation of the list of optimum resources is done at time of submission and the scheduler assigns the *'most optimum'* allocation to the job. The most optimum allocation is the resource allocation will the smallest expected runtime. Referring to the performance curves in Figure 6.4 these optimum allocations can be considered as the lowest point of each graph. If these is space in the system to accept this optimum allocation then the job is pushed to the nodes for execution. This concludes the first stage of moulding.

In the event that the system is busy and the job cannot be executed with optimum allocations the second moulding stage is initiated. The scheduler first evaluates how long will it be till enough resources are free to make the optimal allocation. As the scheduler knows the expected end times for all running jobs this is a trivial exercise of simple lookups. With the expected start time (ST) determined the scheduler adds the time value to the execution duration of the optimal allocation. This now forms the value for total optimal execution time (TOET). Within the second phase the scheduler takes all other allocations smaller than the optimal allocation and determines run times. If any of these *'suboptimal'* allocations gives a time less than TOET the scheduler makes the suboptimal allocation and starts the job. In the event the suboptimal execution time (TSOET) exceeds the TOET the scheduler puts the job in the queue awaiting a change in the system. The system does not look at allocations greater that

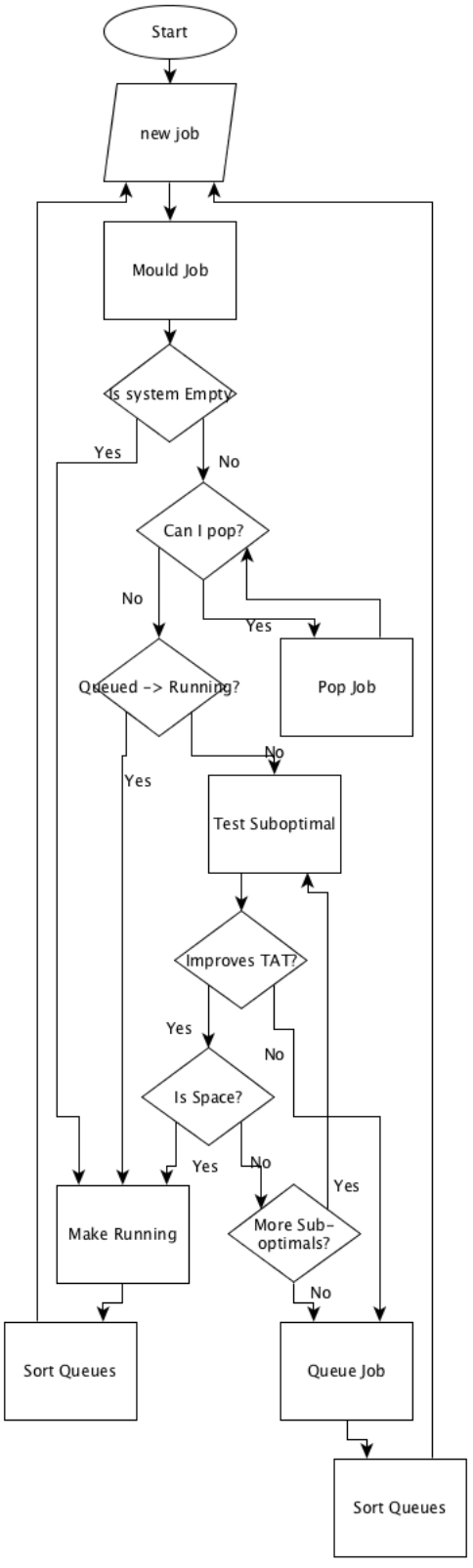

FIGURE 8.2: FCFS Mouldable Scheduler Flowchart

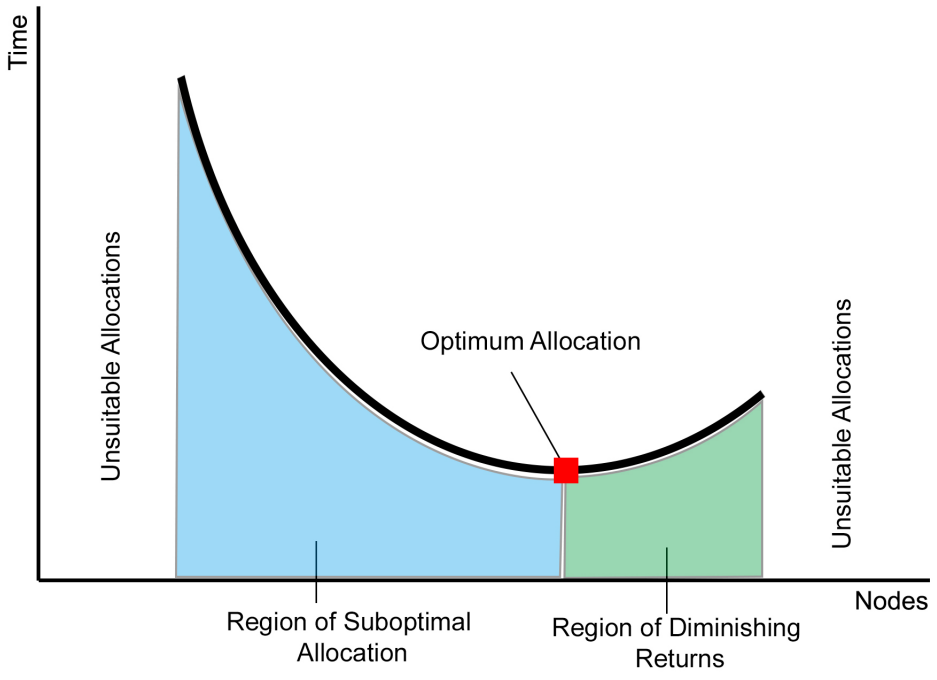

FIGURE 8.3: Suboptimal Decision Making

the optimal as those will likely suffer from diminishing returns. This is depicted in Figure 8.3.

For the FCFS scheduler to save on compute cycles the system does not evaluate the expected wait times for suboptimal allocations. When there is a change in the system status (i.e. a job finishes and vacates a system) the scheduler re-evaluates the job at the top of the queue for optimal and sub-optimal allocations. With jobs waiting in the queue the system will carry out first stage moulds (i.e. determine optimal allocation) and just place the job in the queue behind the waiting jobs.

Once execution begins the rest of the process behaves in the way any scheduler would.

If mouldable scheduling is to be applied to a FCFS backfilling algorithm the primary objective would be to maximise utilisation while ensuring that the TOET of the job at the top of the queue is not affected. So to evaluate the backfill the

scheduler will go through the queued jobs and compare their TOET and TSOET to the ST for the job waiting at the top of the queue. If:

1. the TOET $<$ST then it will run the job optimally, else
2. the TSOET $<$ST then it will run the job sub-optimally, else
3. leave the job in the queue and move on to the next job

In the next section the job submission mechanism is briefly discussed followed by a detailed section of how the scheduler uses the benchmarking information from the ASPP to generate performance characteristic curves.

8.2.1 Submission Protocols

With the Application and System Performance Profiler holding benchmarking and meta information regarding to applications, the scheduler can make allocation decisions based on the application, workload and dataset. All the system requires is for the user to provide this information to the scheduler. For the user this feature is a major shift from the traditional mechanism of submitting jobs. However this new process is more intuitive as the user needs to be concerned only about the parameters that are relevant to their domain. Figure 8.4 illustrates an example of a submission script where a user provides a job name, selects the application and defines relevant parameters.

```
$ jobname=mytest.job
$ application=fluent
$ iteration=1000
$ elements=8000000
$ input=~/simulation/input.trn
```

FIGURE 8.4: Job Submission file for the Mouldable Scheduler System

The parameters are based on the metadata that the ASPP has stored away when benchmarking. The 'jobname' field is used to track this particular job and, on its termination, the heuristics module (shown in Figure 8.1 and discussed in Chapter 11) will use the job name with the run time information to generate heuristic records. The workload manager will then decide on the resource allocation for the job based on the status of the system.

This approach of utilising application specific parameters makes the system more intuitive specially for new users. It makes implementation just as intuitive as now the parameters correlate against the performance characteristics.

It should be noted here that to simplify the development and deployment of the mouldable scheduling algorithm on a real cluster the scheduler currently maintains its own queues and only passes the job that needs to be run down to the TORQUE system. Through future code hardening processes the scheduler will behave as an independant service that will utilise TORQUE's API to manipulate the queues with TORQUE. Further the testing carried out and described for this thesis utilises the Cluster Discrete Event Simulator (CDES) system and is not based on runs on a real system. The real system testing was done as a proof of concept only.

8.2.2 Performance Prediction

After the ASPP has finished benchmarking a system across different dataset and workload sizes the mouldable scheduler has several data points to profile the application on the system. What makes the benchmarks truly effective is that the scheduler can calculate expected performances for datasets or workloads that have not been benchmarked, using interpolation and extrapolation methods. This dynamism is important as it enables the system to deal with a new user or new datasets/workloads.

When a job is submitted to the job management system the performance evaluator module within the scheduler can be invoked. This tool takes as its inputs the user application, dataset size (D_R) and workload (W_R) size information and returns an estimated runtime for the simulation. For the purpose of this thesis the principles of linearisation and weighted averages are utilised to calculate the estimated run times over a range of nodes in the system.

Figure 6.5 gives an example of a linearised surface plot. Any point on that surface would give the corresponding node (x-axis) and time (y-axis) pairs for a given dataset (z-axis). However this is a 4-dimensional problem as workload size also needs to be taken into account. There are 4 cases that the evaluator has to deal with. These are:

1. dataset and workload requested are already benchmarked,
2. dataset requested has not been benchmarked but workload has been,
3. dataset requested has been benchmarked but workload has not been,
4. dataset and workload requested have not been benchmarked.

Case 1: dataset and workload requested are already benchmarked. If the dataset and workloads have already been benchmarked by the ASPP then the evaluator queries the database for the known data-points in that configuration. Using linearisation between the known points the evaluator calculates the estimated performance time for the node/core combinations that do not exist in the database. For an application that has been benchmarked at 4, 8, 16 and 32 cores, the predictor tool can estimate performance times at 12, 20 and 24 cores. If the dataset or workload size exceeds the last known benchmarked data point then the evaluator collects the last two known data points and thus extrapolates the performance. This however is limited to one resource combination in either direction.

Case 2: dataset requested has not been benchmarked but workload has been. If the user requests the estimated performance of an unknown dataset size D_R, the evaluator queries the database for the two known benchmarks immediately adjacent to the requested dataset size (D_0 and D_1) where

$$D_0 < D_R < D_1. \tag{8.1}$$

We estimate that if D_R is closer to D_1 in value, then the performance curve of Time-vs-Nodes for D_R will behave like the curve for D_1. The same can be assumed if D_R is close in value to D_0. Calculating the percentage distance of D_R from D_0 and D_1, the evaluator estimate the predicted run time. This is shown in equation (8.2).

$$f(T_{D_R}) = f(T_{D_0}) + (\frac{D_R - D_0}{D_1 - D_0})(f(T_{D_1}) - f(T_{D_0})) \tag{8.2}$$

where $f(T_{D_R})$, $f(T_{D_0})$ and $f(T_{D_1})$ are the range of run times across nodes for D_R, D_0 and D_1 respectively.

Equation (8.2) holds true where ASPP has a node-run time pair for both D_0 and D_1. For node values where the ASPP has only got benchmarks for D_0 equation (8.3) is used. In conditions were the benchmarks exist for D_1 then equation (8.4) is utilised.

$$f(T_{D_R}) = f(T_{D_0}) + (\frac{D_R - D_0}{D_1 - D_0})(f(T_{D_0})) \mid f(T_{D_1}) = \emptyset \tag{8.3}$$

$$f(T_{D_R}) = (\frac{D_R - D_0}{D_1 - D_0})(f(T_{D_1})) \mid f(T_{D_0}) = \emptyset \tag{8.4}$$

With f(T_{D_R}) evaluated for all possible node-vs-time value pairs, the performance evaluator uses the method outlined in Case 1 to interpolate or extrapolate all required values.

Case 3: dataset requested has been benchmarked but workload has not been, In this case the users has requested a workload that has not been benchmarked W_R, the evaluator queries the database for the two known benchmarks immediately adjacent to the requested workload size (W_0 and W_1) where:

$$W_0 < W_R < W_1. \tag{8.5}$$

Similar to Case 2, case three uses similar equations substituting unknown datasets with unknown workloads. With the node-run time pairs determined for f(T_{W_R}), by the equations (8.6), (8.7) and (8.8), the methods described in Case 1 are used to interpolate or extrapolate the intermediary values.

$$f(T_{W_R}) = f(T_{W_0}) + (\frac{W_R - W_0}{W_1 - W_0})(f(T_{W_1}) - f(T_{W_0})) \tag{8.6}$$

$$f(T_{W_R}) = f(T_{W_0}) + (\frac{W_R - W_0}{W_1 - W_0})(f(T_{W_0})) \mid f(T_{W_1}) = \emptyset \tag{8.7}$$

$$f(T_{W_R}) = (\frac{W_R - W_0}{W_1 - W_0})(f(T_{W_1})) \mid f(T_{W_0}) = \emptyset \tag{8.8}$$

Case 4: dataset and workload requested have not been benchmarked. While in Cases 2 and 3 we have transposed the two dimensional lines from Case 1 into a three dimensional space, Case 4 becomes a four dimensional problem. In Cases 2 and 3 the dataset size and the workload size formed the Z-axis respectively and a surface was modelled. To simplify the problem in Case 4 we break the problem down into two three dimensional spaces. In the first instance we evaluate the problem with workload in the Z axis and then pass the

Cores	Predicted	Actual	Variance
6	400394	394496	1.47%
10	121508	120030	1.21%

TABLE 8.1: DL_POLY MEDIUM benchmarks interpolated for 6 and 10 cores

results to the second three dimensional space with dataset size being the third dimension.

To accomplish this we first query the database for those values of D_0 and D_1 that meet the conditions in equation (8.1). For each of the values of D_0 and D_1 we query the database for the W_0 and W_1 values that meet equation (8.5). This gives us four ranges in total. There is a W_0 and W_1 for D_0, and a W_0 and W_1 for D_1. Using equations (8.6), (8.7) and (8.8), a new range of values for $f(T_{D_0})$ and $f(T_{D_1})$ are generated. So from four performance profiles we are reduced to two. Then using equations (8.2), (8.3) and (8.4) these two performance profiles are reduced to a single profile given by $f(T_{WD_R})$ that meets the users requested D_R and W_R values.

8.2.2.1 Testing Performance Predictions

To test the accuracy of the Evaluator and the efficacy of the ASPP DL_POLY and ANSYS Fluent datasets have been used as discussed in the Chapter 6. Three tests were carried out to analyse the accuracy of the profiler.

The first test was to analyse the accuracy of the profiler at interpolating wall-time values for resource combinations not previously benchmarked. This was done by using the MEDIUM DL_POLY dataset and workload, running the simulation over 6 and 10 cores. The results for this can be seen in Table 8.1

The second test was to assess the interpolation between two sets of benchmarks. To do this the evaluator was tested using an ANSYS Fluent CFD model

Cores	Predicted	Actual	Variance
8	19463	18250	6.23%
12	12583	12341	1.92%
14	8512	8930	-4.91%
16	12083	12046	0.31%

TABLE 8.2: ANSYS Fluent benchmarks with 12M elements

with 12M elements. This model fits within the VLARGE and VVLARGE benchmarks. The evaluator calculates four points of performance. These are over 8, 12, 14 and 16 cores. In the case of the 12 and 16 core calculations the evaluator has a reference point from both the VLARGE and VVLARGE benchmarks. For 8 cores it has only the VLARGE benchmarking data and for 14 cores it only has the VVLARGE benchmarking data. The results can be seen in Table 8.2. The results show that the performance profiler tends to over estimate predicted run times. With only one case where the predicted time was less than the actual run time (indicated by the negative sign). In situations where the Predictor tool is working with two coordinates the results are more accurate. The variance of the results where two reference benchmarks exist, is less that 2%. Where the Predictor tool needs to extrapolate using just one reference point the accuracy decreases.

The third test was to extrapolate an existing benchmark line. This was tested using both DL_POLY and Fluent datasets. In this case due to the linearisation the evaluator grossly over estimates the run times. In some cases this led to a variance of over 50%. This can be overcome in future by using curve fitting techniques rather than linearisation. An alternate solution would be to benchmark the maxima and minima of the system capacity, creating hard limits and clearly defining the available number of cores. This would provide all the boundary conditions and as the previous two tests have shown, when interpolating the performance, the evaluator is accurate to more than 93%.

8.3 Testing Mouldable Scheduler

To test the mouldable scheduler the workload trace of April 2013 was utilised. Described in detail previously (Chapter 5.4), a total of 1150 parallel jobs revolving around 4 users were evaluated. Two of the users were using ANSYS Fluent for Computational Fluid Dynamics (CFD), while the remaining two were using DL_POLY for Molecular Dynamics (MD). The four researchers engaged with the development of the mouldable scheduler and passed on some information relating to workload and dataset sizes.

Using the CDES system the trace log from the Eridani cluster is evaluated. Similar to Srinivasan et al. the trace log had to be manipulated to be able to carry out the testing experiments. Initially, the trace log was augmented incorporate the job related metadata. Tracking job IDs and job names, and with researcher input *application name*, *workload size* (in meta format e.g. iterations) and *dataset size* (in meta format e.g. atoms) was appended to the end of each job log. This augmentation was carried out as it was believed that the results would be more accurate if no synthesis of the actual torque logs takes place and essentially a *real* workload is executed.. However upon further considerations a key flaw became apparent and the efficacy of the results could be questioned.

A predicted run time may or may not be actual run time. That is just because a scheduler believes a job will take (e.g.) 24 minutes to execute it does not mean that on an actual system the job is guaranteed to end in that time. If a scheduler would force a job to end based on its own predictions this would cause great hinderance to the user. Unfortunately in a simulated environment like CDES it is nearly impossible to model such behaviour. Uncertainties could be programmed into the simulator however this would relate to very random results with no reproducibility. HPC simulation generally revolves around the

Type	Description	Truncated Logs
Real Logs	Unaltered logs from Torque	Appendix A.1
Moulded Log	Augmented Logs with job meta information	Appendix A.2
Normalised Log	Normalised logs with new durations and meta information	Appendix A.3

TABLE 8.3: Input Data for the Mouldable Scheduler

concept that if all things remain equal the trace logs can be used to reproduce the system behaviour.

While in the trace log two jobs with identical applications, workloads and datasets can have execution durations that differ, in the CDES moulded environment this variance will be lost as the scheduler will force the job to end in the predicted time. To mitigate this effect during the testing an intermediate simulation was carried out. Aside from the real trace log, a normalised trace log was created. The normalisation processes involved using the evaluator module (described above in Section 8.2.2) and the user provided resource constraints to synthesise new logs with revised durations for each job. Based on the moulding information and the nodes/cores requested the evaluator module provided expected runtimes as calculated from the benchmarks. Overall the total duration that any users jobs took adds up to the same amount of time in the Real and Normalised logs. The different logs are shown in Table 8.3.

For the purposes of testing the full workload of the month is not utilised. The trace log is stripped of all serial jobs, parallel jobs where the applications were not benchmarked, and those jobs for which meta information was not known. The remaining workload was tested against a 16 node cluster rather than the standard cluster size of 32-37 nodes.

Figures 8.5, 8.6, 8.7, shows the affect of normalisation on the real workload

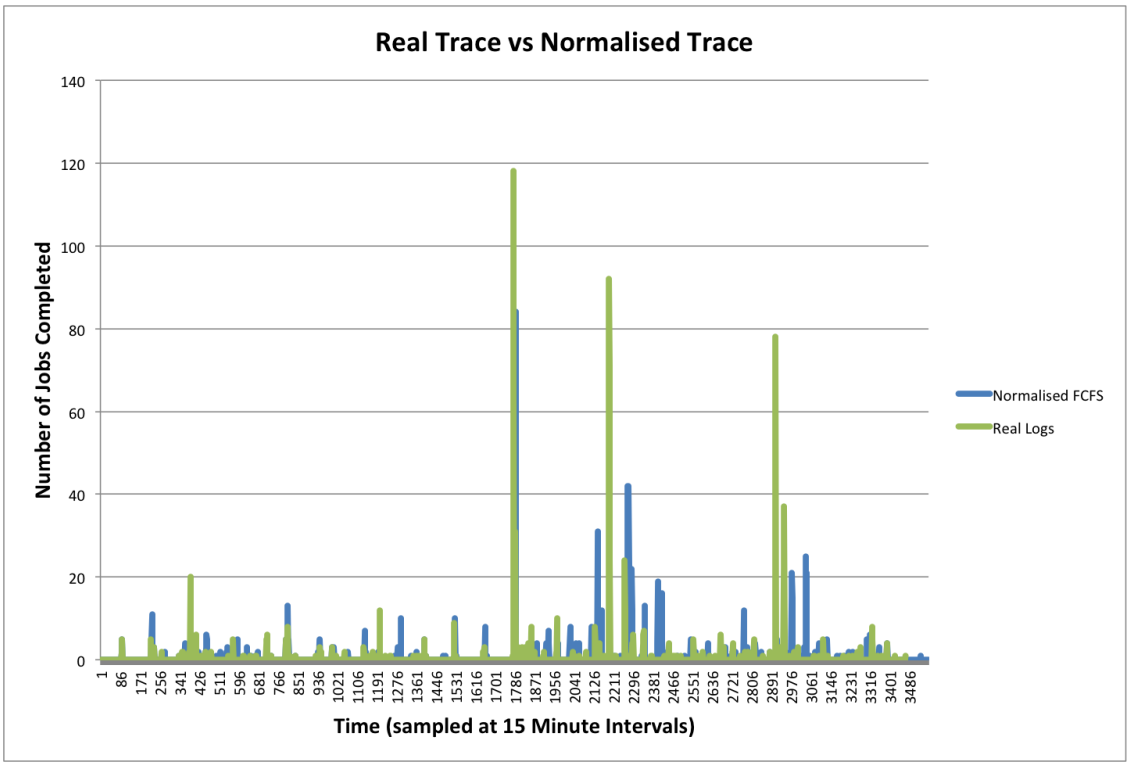

FIGURE 8.5: Real Data vs Normalised Data FCFS [15min Intervals]

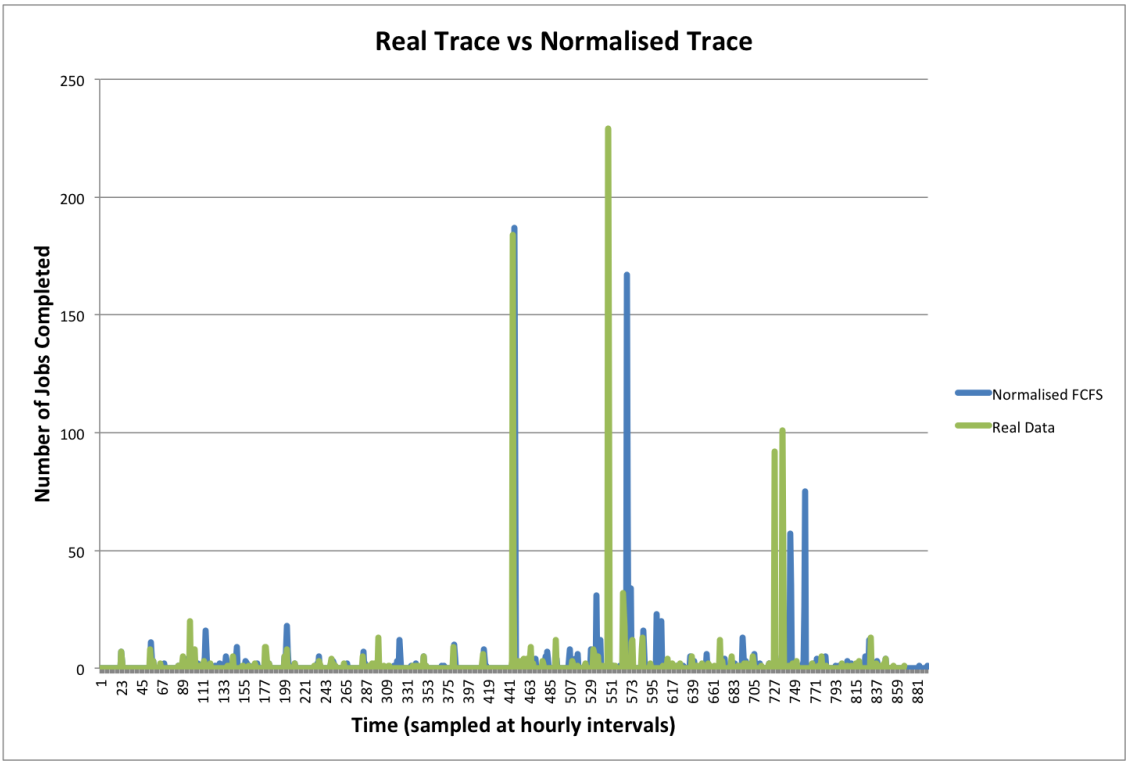

FIGURE 8.6: Real Data vs Normalised Data FCFS [1hour Intervals]

broken down over a 15minute, hourly and daily completion rate. On a day-to-day analysis normalisation only slightly affected the logs. There are only two

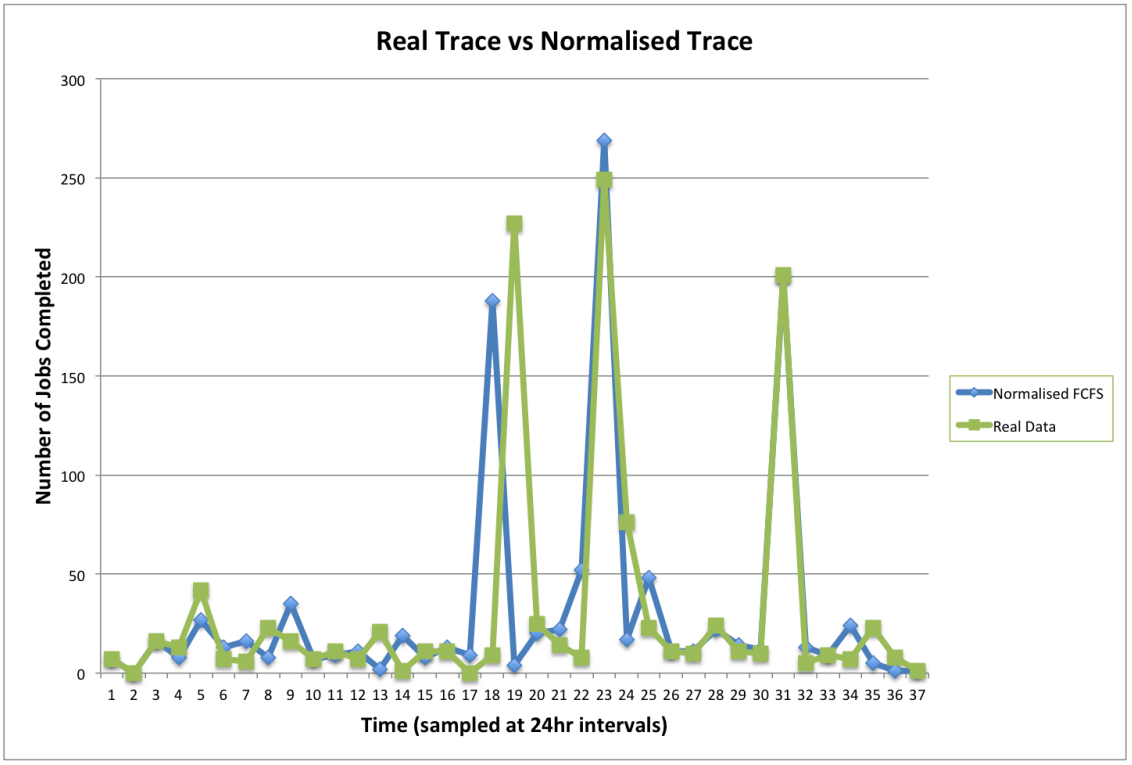

FIGURE 8.7: Real Data vs Normalised Data FCFS [24hour Intervals]

spikes where the normalised workload finishes faster than the real data (on day 17 vs 19 and on day 34 vs 35). This would be due to earlier jobs being mostly rounded down. Obviously rounding job times will cause greater variance when viewed at the 15 minute granularity. Both workloads completed execution on the same day, albeit several hours apart. Both logs were executed against the FCFS with backfilling algorithm outlined in Chapter 7.

With the normalised workload in place and its similar execution profile to the real benchmarks the same log file was used to test the mouldable scheduler. Figure 8.8, 8.9, 8.10 show how the mouldable scheduler performed as apposed to the normalised logs using the FCFS algorithm. In total 504 jobs (˜44%) were moulded to different resource allocations at submit time. Of the 504 jobs that were pre-moulded a majority of them (˜77%) we given allocations greater than those provided by the user. This is a significant result even though at first appearance it would seem unintuitive. Most users would select 2 nodes over 3-4

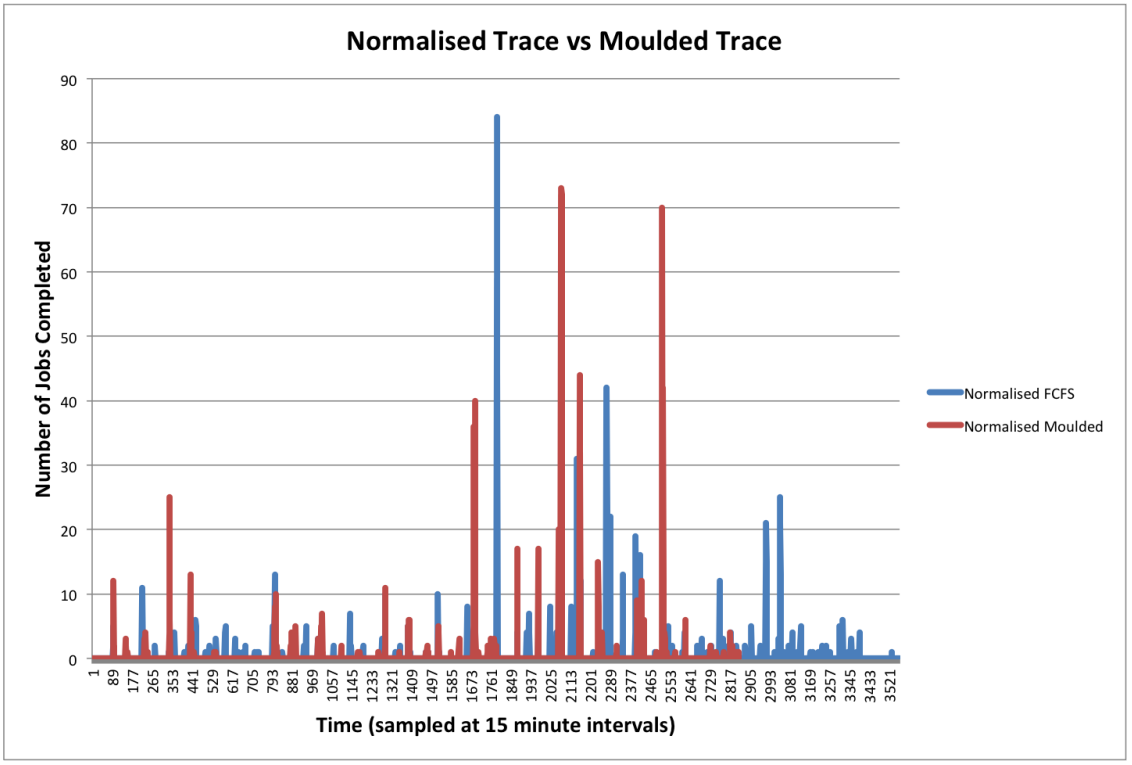

FIGURE 8.8: Normalised Data FCFS vs Mouldable [15min Intervals]

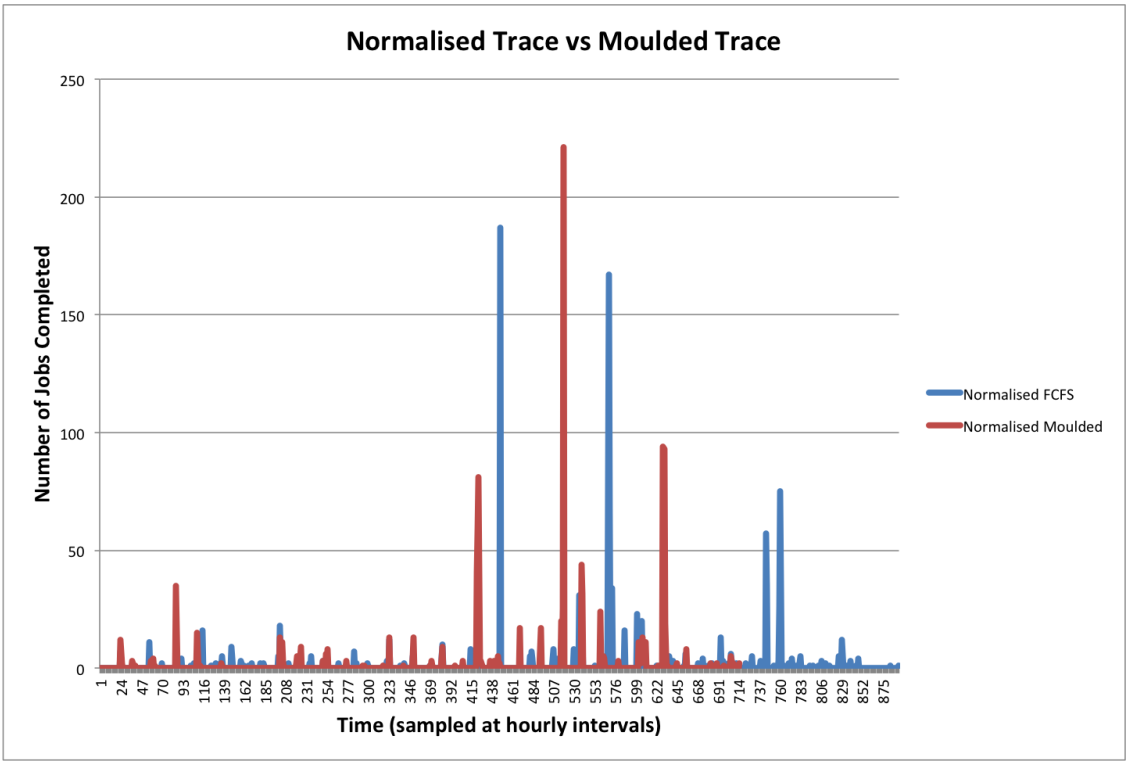

FIGURE 8.9: Normalised Data FCFS vs Mouldable [1hour Intervals]

node allocations because they would want better throughput. However these results show that by giving optimal allocations overall the system throughput

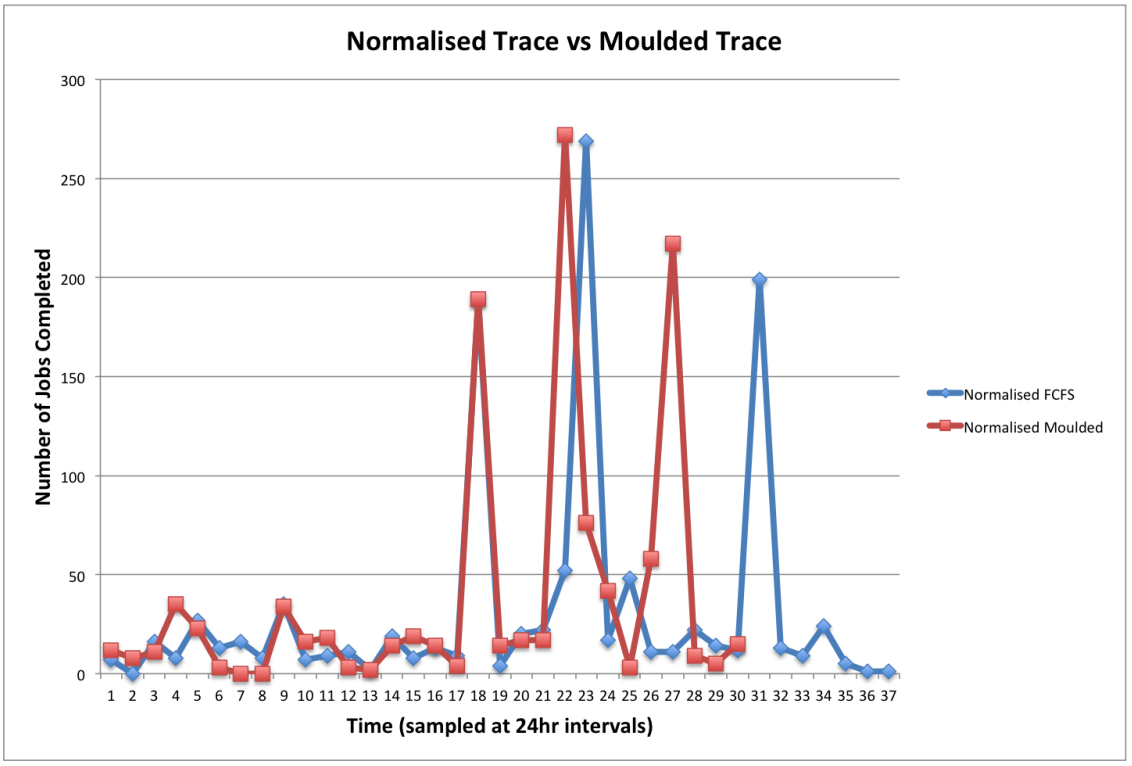

FIGURE 8.10: Normalised Data FCFS vs Mouldable [24hour Intervals]

increases.

During the execution 34 (˜3%) jobs were sub optimally moulded to improve system throughput. The large 8 node jobs from the molecular dynamics group were all run sub-optimally. In the FCFS with Backfilling situation these jobs had to wait till most of the smaller jobs had passed through before enough resources became available. By moulding them to 5 and 6 node allocations the mouldable scheduler was able to get the jobs running quicker, even though the jobs ran longer and processed the queue faster. From the users perspective the turn-around-time improved, and from the systems perspective the system was utilised more efficiently.

Overall the mouldable scheduler was able to complete the entire workload 8 days faster than its non-moulding counter part. That relates to being 1.28 times faster over the month.

8.4 Summary

As shown in the previous section the mouldable scheduler is able to dramatically improve the turn around time of jobs in the system. As a significant number of jobs were moulded at submit time and not at run time, these results would suggest that system performances will improve if users selected optimum allocations rather than choosing the smallest possible allocations to improve perceived throughput. This was a small subsection of jobs on a small system. When scaled up these optimal allocations would greatly improve system throughput.

Due to the lack of real world production systems able to do mouldable scheduling this research, like others before it, has had to rely on augmented and synthesised logs as well as a simulator. Efforts have been taken to ensure accuracy. The augmentation of the trace logs does not affect the performance but the normalisation, to create synthesised logs does slightly alter the throughput of the system.

To algorithm to mould jobs is computationally expensive. While the cost does not become apparent in a first come first served scheduler, if any out-of-order execution is attempted the number of lookups and calculations required would dramatically increased. In its current implementation it is strictly suited for a FCFS environment. Section 9.2 addresses a novel method to reduce the computational complexity and make the system more efficient in its decision making.

Overall the mouldable scheduler is able to out perform FCFS and FCFS with backfilling, giving significant improvement to turn around time. The system also adjusted for bad-put which to administrators was an obvious problem with jobs on the system. What was surprising however that part of the bad-put that the

system adjusted for was advance (power) users under-booking resources. Perhaps with a full workload of serial and other parallel jobs, the scheduler would attempt sub-optimal allocations for the power users jobs.

This system of dynamically determining appropriate resources is a significant change which will aid in scaling HPC workloads into Cloud computing infrastrucutres (shown in Section 9.3).

Chapter 9

Scheduling Paradigms

9.1 Introduction

With the creation of the mouldable scheduler (as described in Chapter 8) several avenues of research opened up. In particular with the hype cycle and maturation of cloud computing reaching its zenith the opportunity to leverage the scheduler became apparent. In this Chapter the research and implementation of surge computing is outlined (Section 9.3). The implementation described here utilises standard scheduling techniques and the impact of the mouldable scheduler in this context is presented. Section 9.2 outlines a method to speed up the decision making processes of the mouldable scheduler to make the system computationally inexpensive.

9.2 Fuzzification of the Workload Manager

9.2.1 Background

Mouldable Scheduling strategies are complex and require system calls, comparisons and modelling predictions. These are computationally expensive tasks which will not scale well under different scheduling algorithms. In Section 8.2 the mouldable scheduler is implemented in a First Come First Served (FCFS) environment. While a backfilling implementation is briefly described, implementation of such a system has many challenges.

Within a FCFS environment only the job waiting at the top of the queue needs to be moulded to space available in the system. Depending on the size of the system and possible allocations for any job, the number of comparisons and calculations required can number in the tens. The problem itself can be classified as being O(n) as:

$$TotalCalculations = n - PossibleAllocations + n - EndTimeCalculations \quad (9.1)$$

However if the mouldable scheduling strategy is applied to a FCFS with Backfilling algorithm, then the complexity greatly increases. The problem escalates to $O(n^2)$. For every iteration of moulding the system will need to work its way down a list of jobs attempting to backfill. This makes the problem computationally very expensive.

In order to reduce some of the complexity it is proposed that fuzzy logic techniques are utilised to make moulding decisions. Fuzzy Logic is a technique

within Artificial Intelligence for approximate reasoning. For a fuzzy implementation of the moulding decisions the characteristics of linguistic variables and fuzzy rules are utilised. Linguistic variables are representation of state or value using words instead of numbers (e.g. big,small,long,short). In a fuzzy system the rules are used to make the decisions (Zadeh, 1994; L.-X. Wang, 1999).

From an operators perspective certain scheduling decisions appear obvious. For example *"the system is very busy and this is a large job so there is no point trying to mould it."* A standard decision system will not be able to reduce the problems to these two conditions. With fuzzy logic the two variables can be linked using certain rules (see Figure 9.1). Under the implementation described in Section 8.2 the system will verify all possible allocations against the system status before ruling that a job can not be moulded.

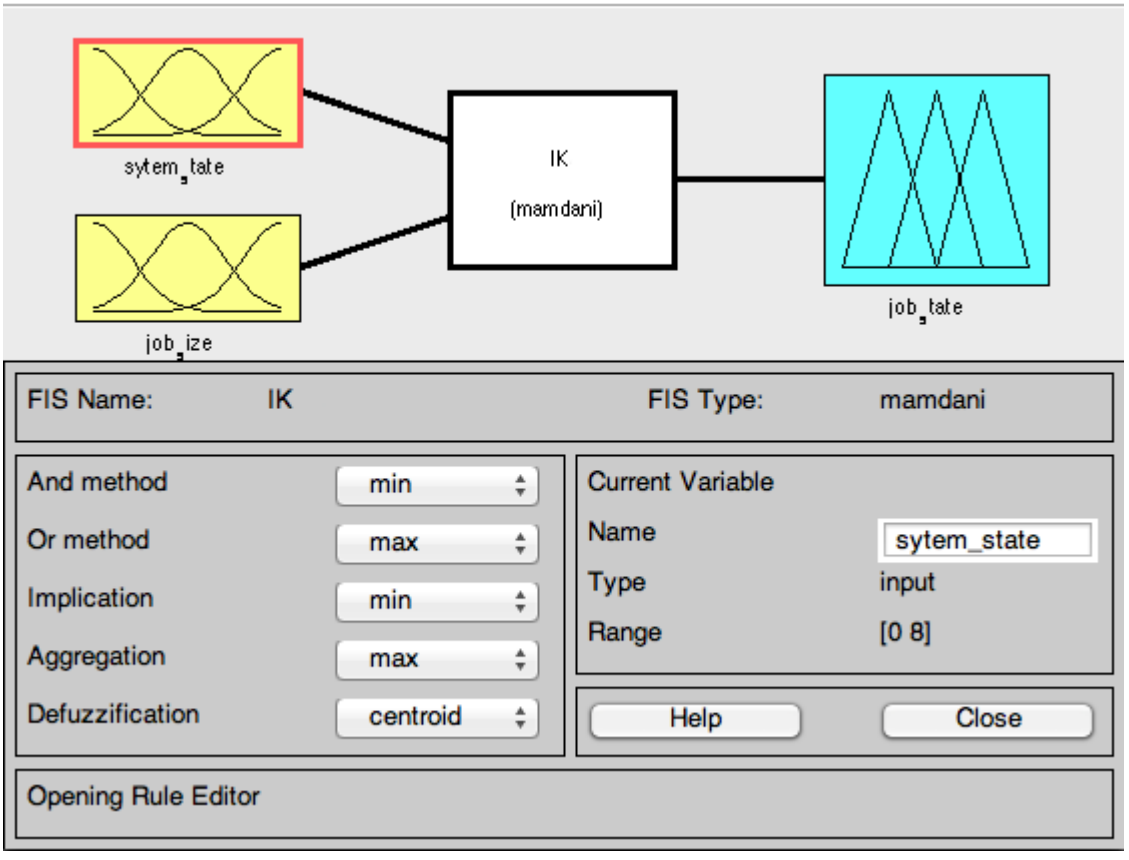

FIGURE 9.1: The Moulding Decision in a Fuzzy Engine

Job Size	System Status	Scheduling Decision
SMALL	FREE	Run
SMALL	NOT BUSY	Run
SMALL	**AVERAGE**	**Attempt Mould**
SMALL	**BUSY**	**Attempt Mould**
SMALL	FULL	Wait
MEDIUM	FREE	Run
MEDIUM	**NOT BUSY**	**Attempt Mould**
MEDIUM	**AVERAGE**	**Attempt Mould**
MEDIUM	**BUSY**	**Attempt Mould**
MEDIUM	FULL	Wait
LARGE	FREE	Run
LARGE	**NOT BUSY**	**Attempt Mould**
LARGE	**AVERAGE**	**Attempt Mould**
LARGE	BUSY	WAIT
LARGE	FULL	Wait

TABLE 9.1: Linguistic Variable Value Pairs for Mouldable Scheduling

9.2.2 Implementation

The terms ”system is very busy” and ”this is a large job” can be described as linguistic variable - value pair. There are several states the system can be in like free or idle; not busy; average load; busy; and full. Similarly jobs can be of different sizes e.g. small, medium and large. Table 9.1 shows the different linguistic variables and the scheduling decisions to be made.

System operators through their experience can quantify linguistic variables such as busy system and medium sized job. A small job can be a job requiring $<$25% of the system, a medium job between 15% and 55%, and a large job $>$40%. Similarly the system is free when there is 0% load and full with 100% load. Not busy can be quanitied as $<$30% load, average between 20% and 55%, and busy would be between 50% and 99%. This linguistic variables are hard and fast conditions and the actual state can in fact be classed in 2 or more ways.

As shown in Table 9.1 moulding is only attempted in 7 out of 15 possible states. Since the mouldable scheduler does not attempt a mould if the system is free

and just runs the optimum allocation the total states where moulding is considered in a backfill are 12. With fuzzy logic techniques reducing that to 7 mould states it means processing requirements in over 40% of conditions is avoided. Figure 9.2 shows these rules applied in a Fuzzy Engine.

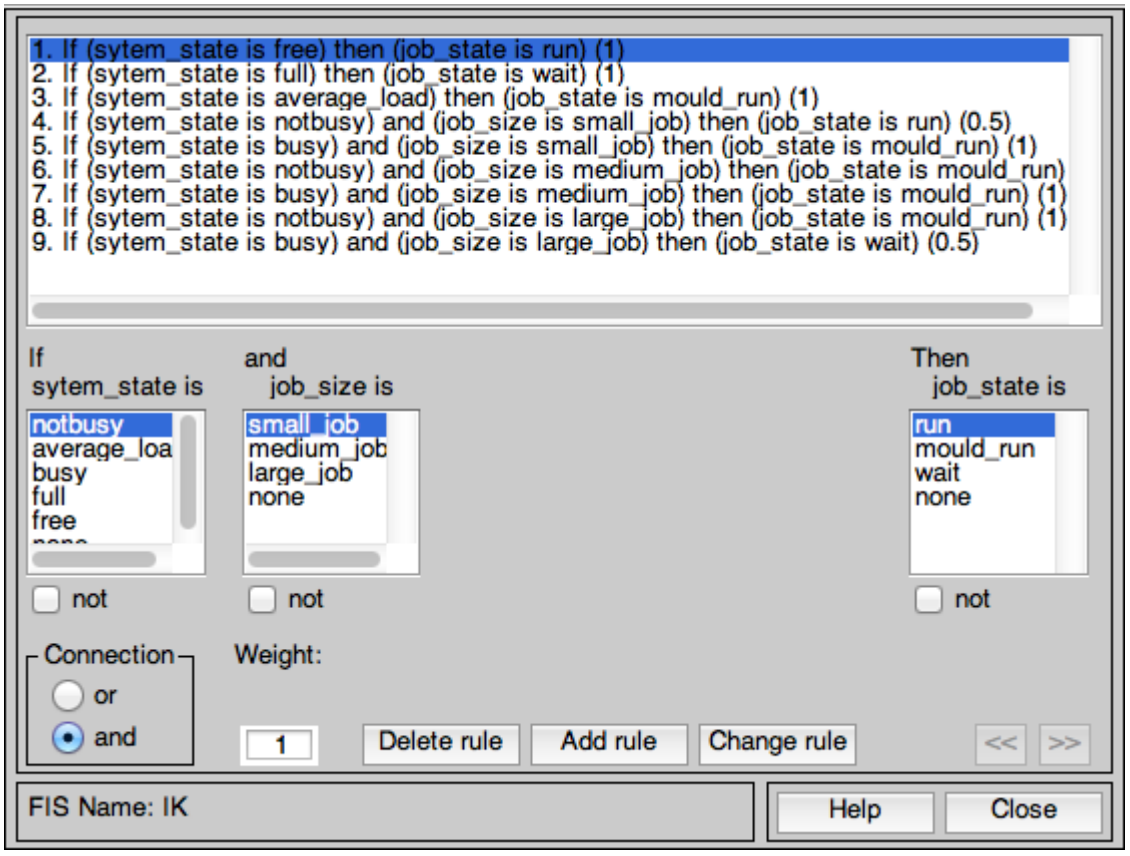

FIGURE 9.2: Rule Sets Applied Using Fuzzy Logic

These linguistic variables and corresponding membership functions have been implemented in MATLAB. Figure 9.3 shows the membership functions for the system conditions and Figure 9.4 shows the membership functions for the job sizes. The overall system is shown in Figure 9.5.

Using the Mamdani Inference method the the rule sets are applied to the linguistic variables to make decisions based on MIN-MAX conditions. Certain rules have a higher weight as shown in Figure 9.2. In Figure 9.6 the resulting surface plot of the actions is shown. Regions in Blue are where run conditions are met

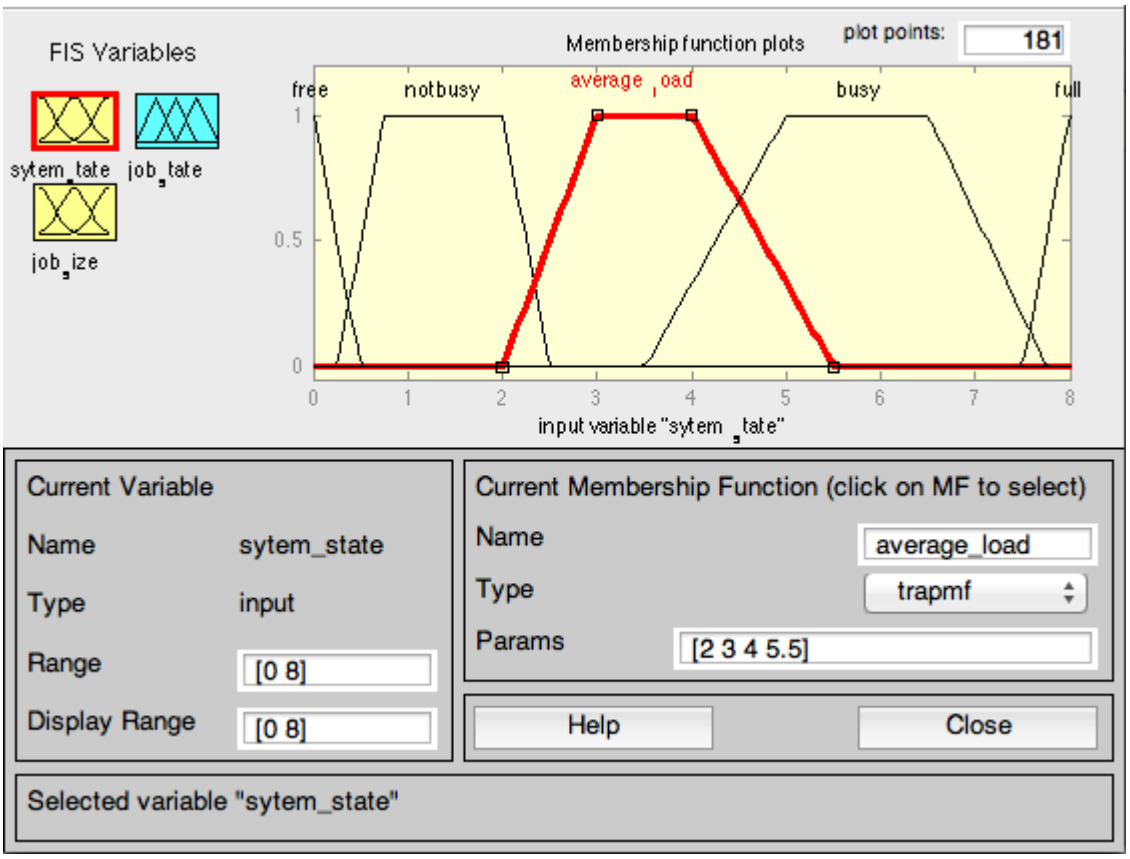

FIGURE 9.3: Linguistic representation of System State

and Yellow is where wait conditions are met. In other regions the system will attempt to mould the jobs.

Figure 9.7 depicts a situation where the system can be classed as having an ”average load” bordering on being ”busy” (column 1). The Job can be classified as small or medium (column 2). This is because the job size lies just as the ”small” size begins to taper off and where the ”medium” size begins to plateau. The third column shows that 3 different rules get triggered, Rules 3, 5 and 7. The fuzzy engines output is to mould the job.

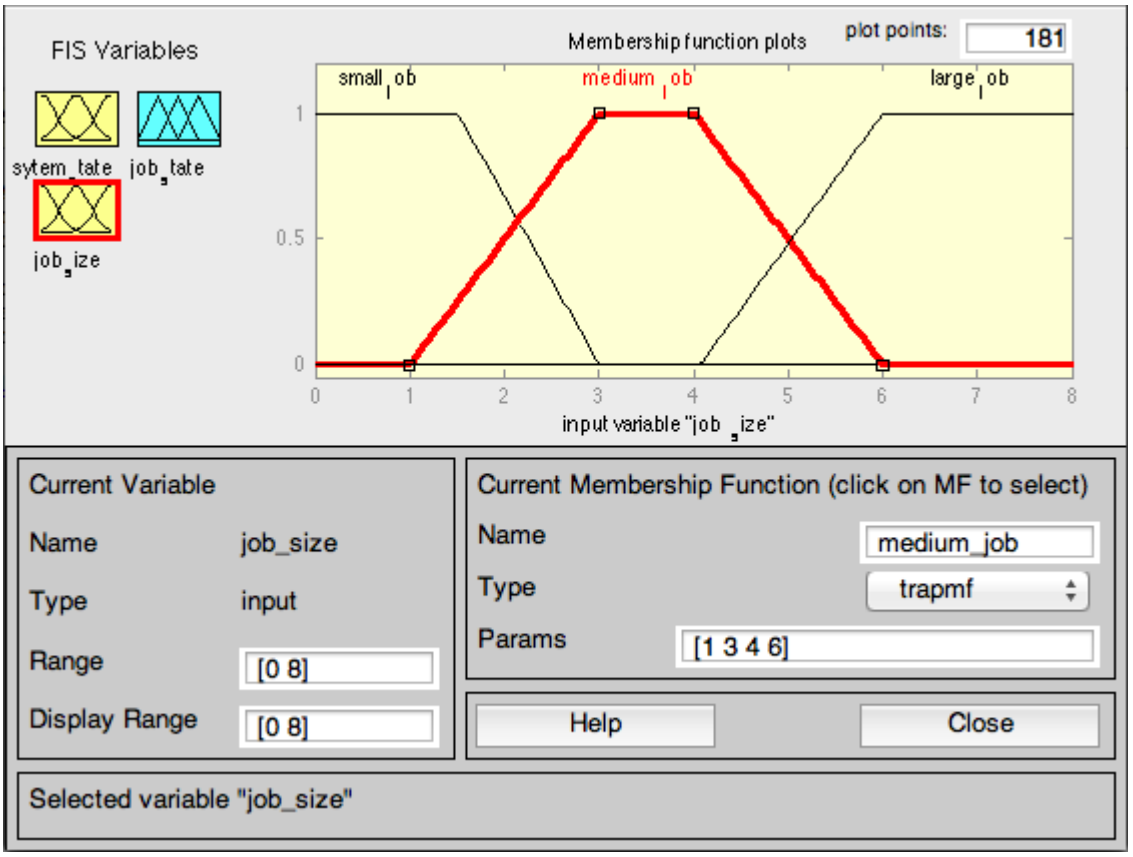

FIGURE 9.4: Linguistic representation of Job Size

9.2.3 Discussion

The Fuzzy Engine described in this section can greatly reduce the processing overheads that will be introduced in out-of-order mouldable scheduling algorithms. A second engine can also be implemented to further reduce the processing required to verify the performance of sub-optimal allocations. The availability of Fuzzy Logic toolboxes in the major programming languages means that integration with the Cluster Discrete Event Simulator (CDES) system and traditional HPC schedulers is possible.

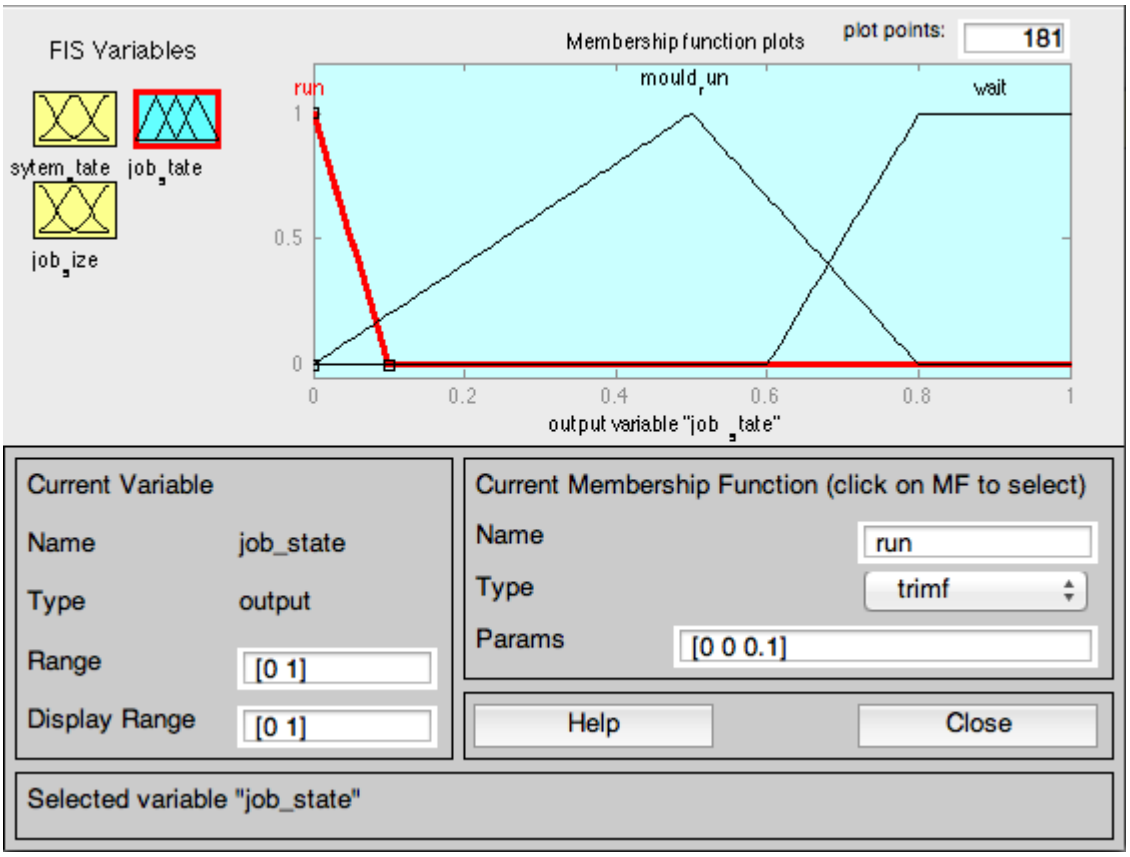

FIGURE 9.5: Moulding Decisions based on Fuzzy Logic

9.3 Surge Computing: Elasticity in Scheduling

9.3.1 Motivation

The University of Huddersfield's Research Computing Infrastructure (RCI) comprises of several High Performance Computing (HPC) clusters. They differ from each other by core speed, interconnect speed, co-processor availability and memory configurations. As discussed in Chapter 5 on average most systems have a 2GB RAM per core core memory configuration and the maximum available memory configuration is 4GB RAM per core. At a node level memory is limited to 16GB RAM per node.

Where surge computing was identified as a solution for internal HPC needs was under the following scenario. A researcher using a commercial *NIX based

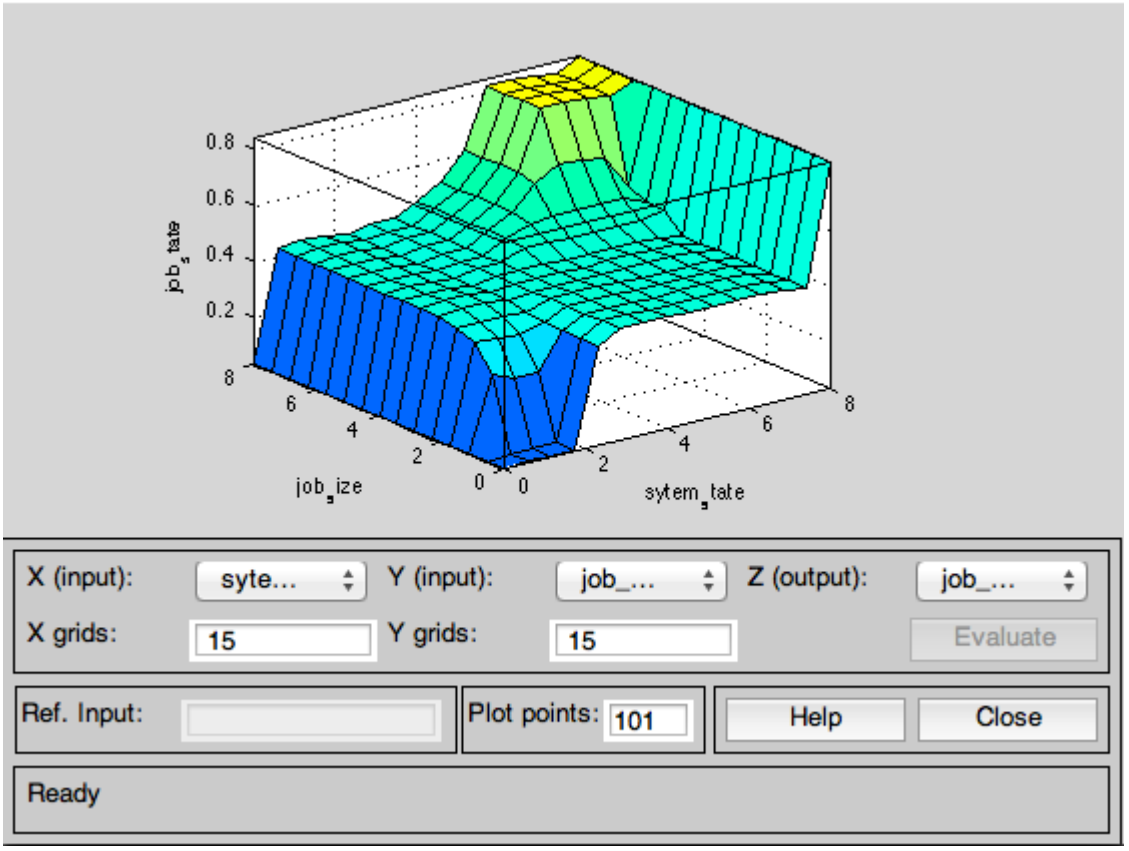

FIGURE 9.6: Surface Plot of Resultant Ruleset

rendering package needed to generate high resolution visualisations of blood flow. Under most conditions the rendering package divides the frames between nodes and speeds up the render time. However, for the resolution required for the final product each frame required more that 16GB of RAM at node level. Due to the commercial nature of the software package, the application can not be ported to other architectures (like POWER or BlueGenes available at the STFC-Daresbury), nor can it be deployed on a partner institutions HPC system. Before HPC deployment of this package, researchers bought 8GB RAM workstations, followed by upgrades to 16GB and then finally a single 32GB RAM workstation. The purchasing power of the research group has quite clearly been affected by a single computational problem.

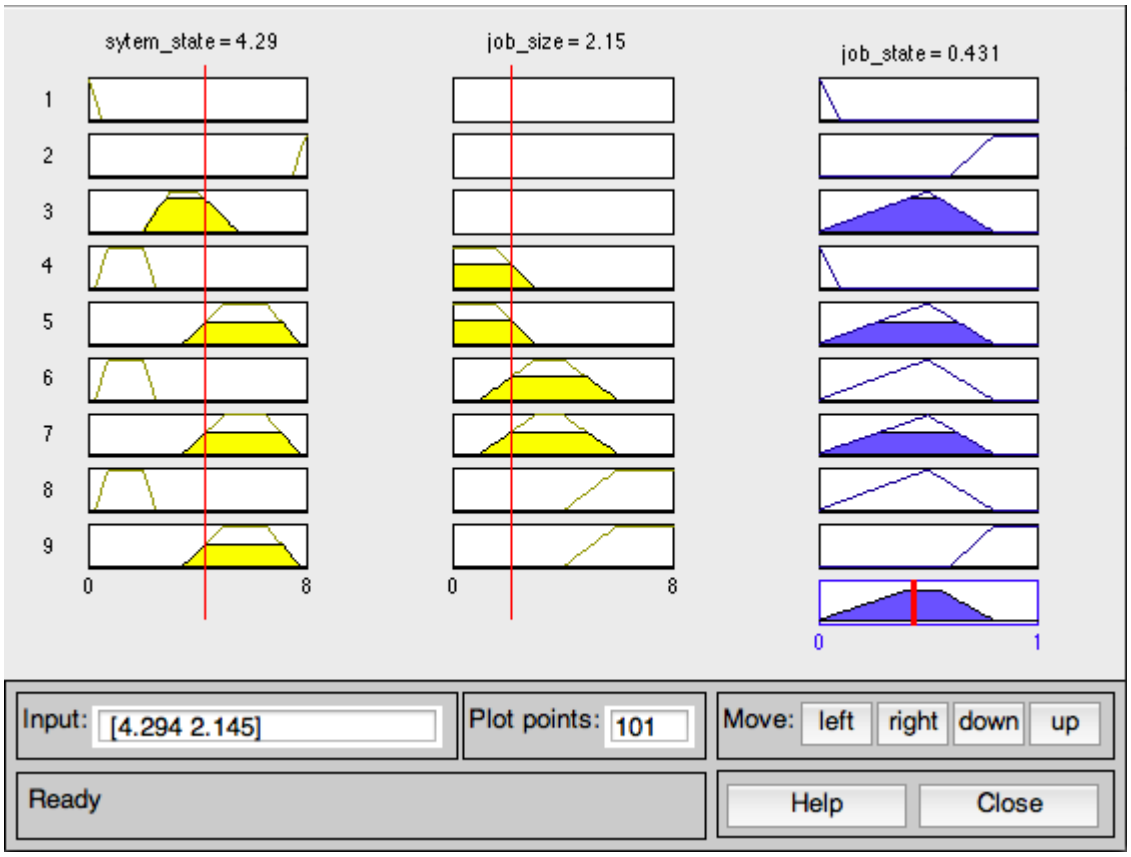

FIGURE 9.7: Example Case of Fuzzy Scheduling

The purchased workstations are power hungry in an idle state and, for the majority of the time, are used for standard office work! Provisioning for such applications within a cloud allows for delivering different hardware configurations that the researcher could not possibly get under their desk. The key however is to remove the complexity of the cloud and to seamlessly integrate this system within the existing HPC infrastructure so that the researchers workflow is not significantly disrupted.

The University's provision for cloud computing is build upon the cloud middleware tool known as Openstack. Openstack is an Amazon EC2 compatible middleware that is popular for both private and public cloud implementations.

9.3.2 Implementation

In the initial development, a wrapper script for PBS/TORQUE was designed. Using Python extensions for Keystone, and a MySQL database to hold configurations, the wrapper script decides if a job should be directly submitted to the underlying HPC system or surged to the prototype cloud system.

9.3.2.1 Decision Metrics

When a job is submitted to the cluster, the wrapper intercepts the job file. If the requested resources (in the job file) match those resources that can be found within the hardware scope (of the traditional cluster), then the job carries on through the system as normal. If the requested resources exceed what is available as bare metal, then the wrapper begins to provision a node in the cloud. Using information defined in the configuration database, along with the status of the load on the cloud, decisions are taken regarding the provisioning. A flow chart showing the steps is shown in Figure 9.8.

If the requested resources can fit on pre-defined cloud limits, then the wrapper initiates a compute node instance within the cloud and submits the job to TORQUE, along with a flag that ensures that this job will be routed to the new cloud instance. If the requested resources exceed the limits of the local cloud service, and the user/administrator has configured credentials to surge to a public cloud, then the VM is instantiated within the public cloud. In the event that there are no public cloud credentials, the job is returned to the user with an error message.

To maximise the utilisation efficiency of the private cloud, the wrapper is able to create a hardware flavour within cloud. This flexibility is essential in a private cloud setting. Generally provision options on clouds tend to be configured in

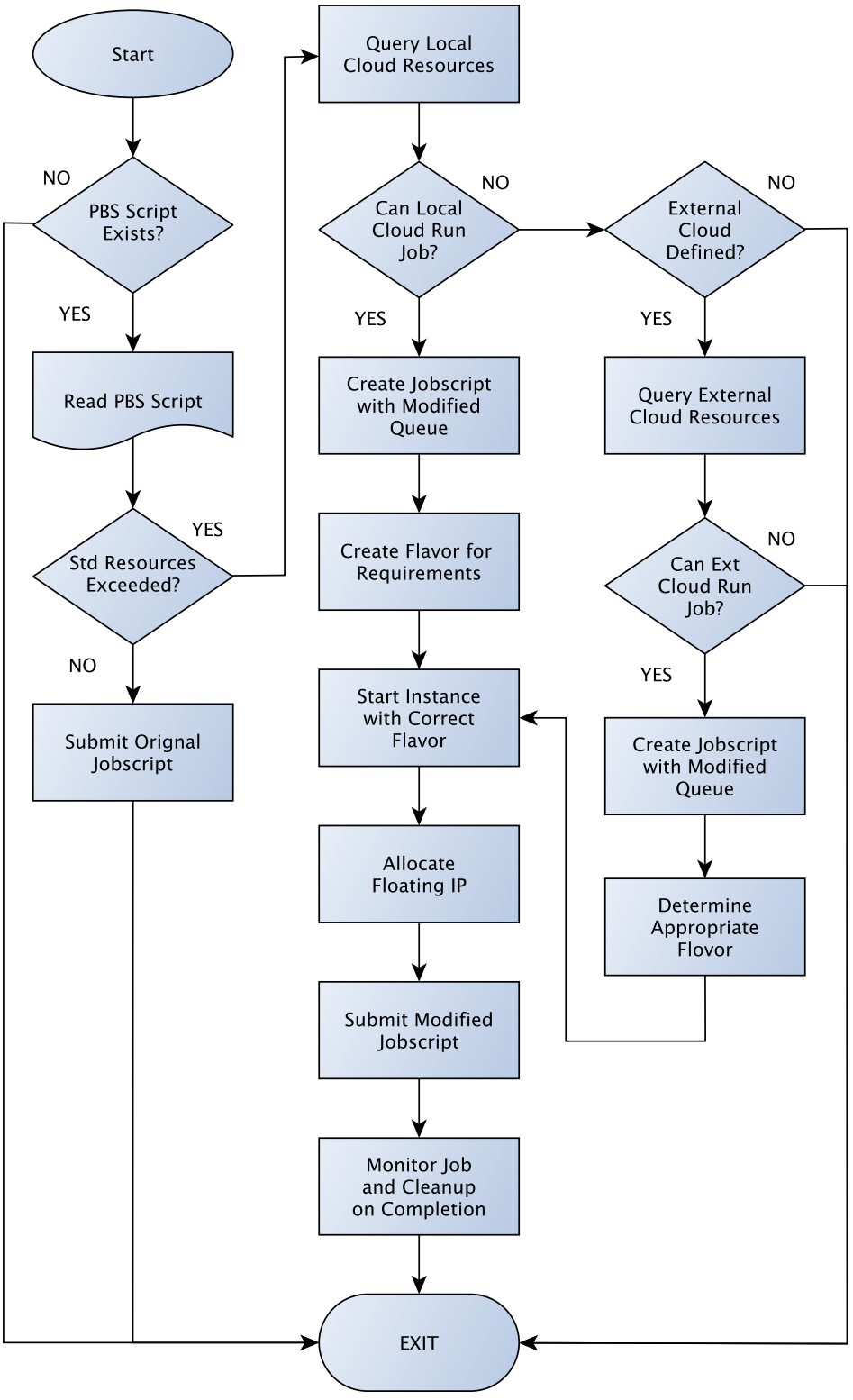

FIGURE 9.8: Flowchart depicting the decision making process in HPC Cloud Surging

binary increments. So after a 16GB RAM flavour the next flavour will have 32GB of RAM. Therefore if a user requests 20GB of RAM for their simulation there will be 12GB of RAM booked and not used. It will also limit the total number of surge instances created.

9.3.2.2 Surge Wrapper for TORQUE

Users on the surge enabled HPC system have a custom script named 'qsub' in their default paths. This qsub is not the standard TORQUE submission executable. The script passes the users submission to the surge wrapper. The surge wrapper parses the arguments supplied by the user to PBS/TORQUE. If a job is submitted with more than the job file argument, the wrapper automatically passes the job down to the scheduler. To invoke the surge a user has to define their job within a PBS job file.

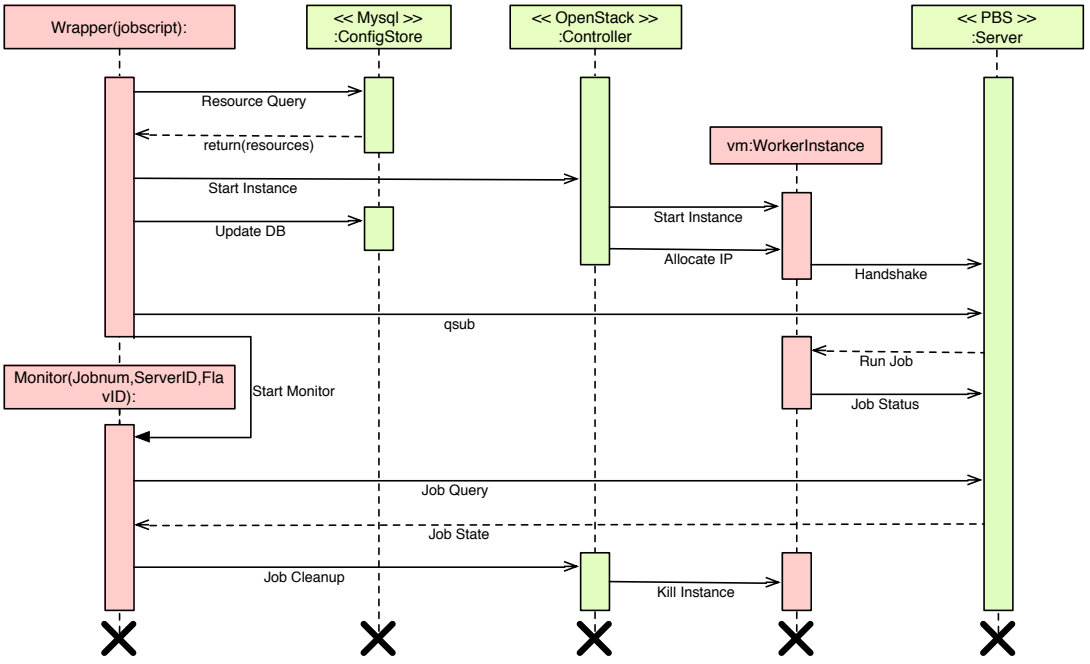

FIGURE 9.9: Control bus depicting the sequence of events in an HPC Surge

Using PBS based functions, the wrapper queries the underlying HPC system to assess the available capabilities. Once it has ascertained that the requested

resources are not natively available via TORQUE, it calls the function to surge the job into the cloud. Using the limits specified by the configuration database, the wrapper then generates a hardware flavour template. This is injected into OpenStack. The configuration database holds information such as: how elastic the system is; the largest flavour instance possible; etc.

With the hardware flavour injected into the system, the wrapper starts a virtual machine instance utilising a pre-configured image. This image mirrors a typical node within the HPC system. Next, the job script is regenerated with a special queue name and handed over to TORQUE. The wrapper also starts a small service to monitor the job's progress.

On the system TORQUE is pre-configured to recognise 'n' number of nodes (where n is equal to the maximum nodes that can be generated in the cloud). These nodes are locked to the special queue. So when the wrapper submits the job with the modified queue, TORQUE waits for the instance in the cloud to start up and then matches the job to the cloud instance. From the TORQUE point of view, a job has arrived which can only be assigned to a particular set of nodes and, one of those nodes has just become available.

Within the cloud, OpenStack generates a new hardware flavour and instantiates an image against this flavour. It then subtracts the requested resources from the pool of resources the surge can request. This is fed back to the surge wrapper to ensure the cloud does not get overly subscribed.

Upon completion of the job, the TORQUE MOM demon becomes idle and, once the job monitoring service detects this, it injects a poison pill to terminate the VM instance. The custom hardware flavour is additionally removed from the OpenStack environment. Within TORQUE, the node becomes offline. Figure 9.9 depicts the sequence of events that take place when a job is surged to the private cloud.

Whether being used in a public or private cloud environment, the systems scalability will always be controlled. In a public cloud the availability of resources purchased (or some form of financial cap) will prevent an indiscriminate number of nodes being spawned. In a private cloud the elasticity itself is limited. OpenStack controls the number of cores and instances a user can run through group policies. In OpenStack parlance, this is know as 'Project Quotas'. In the situation where the surge 'project' on OpenStack is out of resources to spawn a new node, the new job is simply queued. If the new job, e.g. job-B, can run on already spawned virtual machines, then job-B is sent to TORQUE in the modified queue. The monitoring service that is awaiting the completion of already started/running jobs, is passed the new jobs information. Upon completion of the running jobs, the monitor refrains from injecting the poison pill, allowing TORQUE to push job-B to the now idle node. If the spawned instances do not meet the requirements of the new job, then job-B is held in a queue external to TORQUE. Running jobs will terminate as normal, and instances are brought down returning their resources to the quota. The job is then reconsidered for execution when the monitor reports a change to the load on the cloud.

If a requested hardware configuration can never be met, an error is returned to the user. Information pertaining to the maximum number of nodes, smallest configuration of nodes, largest configuration possible, static flavours (if using a public cloud) and credentials, are available to the wrapper from the MySQL configurations database.

9.3.3 Discussion

The HPC surge solution described in this Section is a step towards automating the migration of workloads to the cloud. Using the mouldable scheduler from Chapter 8 and the fuzzy engine described in 9.2 it is envisaged that research

groups may only need a single cluster head node on site and the computation can take place in the cloud. Unlike rigid systems in the cloud the surge element can help the system spawn only those nodes that are required. If this approach to HPC in the Clouds becomes popular, application developers can themselves carry out benchmarks on known Cloud services.

Chapter 10

Conclusion

As Research Computing Systems (RCS) move from single owner internal High Performance Computing (HPC) systems to publicly shared national resources and cloud computing platforms traditional scheduling methods are falling short. These systems are unable to adapt to the dynamic nature of new computing paradigms like cloud computing. By outsourcing the heavy infrastructure, inevitably the in-house system administrators are few and far between. Inexperienced users will not have the safety net usually in place to help them understand how to get the best performance for the systems for them to do their science. There is a real need for an autonomic system to decide the best allocation of resources to provision for a user, based on the application, dataset and workload needs.

Where in-house systems exist the pressure to *go green* has never been higher. Out-of-order executions are the norm to ensure 100% utilisation of the HPC systems. However in these days of fast computers and fast networks holding a users job back because *"it is to big to fit at current load"* is simply unacceptable. As is the money wasted in electricity and technician time due to *"bad-put"* by the users.

Mouldable Scheduling in general and the solutions presented in this thesis in particular are the step changes needed within production environments to address the challenges of these new computing paradigms. The Rule Based Mouldable Workload Manager, the Fuzzy Moulding Engine and the Surge Computing systems are the building blocks towards scheduling in the clouds and towards exa-scale. With these stepping stones there is still much research do be done.

Starting from some basic tools that already existed within HPC systems, benchmarking, and scheduling, this thesis has demonstrated the steps required to create a mouldable scheduler. In the first instance there needs to be a mechanism for a system to understand how an application will perform given a particular dataset and workload. Previous attempts by other investigators either fell short of this goal or utilised cycle counting methods to predict how an algorithm would behave. These methods are system and network agnostic and also do not take into account proprietary optimisations or compiler interventions. The Application and System Performance Profiler (ASPP) presented in this thesis is a step towards not just creating a benchmarking suite but to also profile an application. While it is able to answer the question ”Whats the fastest my application can run?” it also answers the critical question of ”How will my application perform at different resource allocations”.

Data from the ASPP is then fed back to the proposed mouldable scheduler. For the first time a scheduler is able to determine how the ”real” application performs with the users ”real” problem set. The user too no longer needs to worry about processor or memory allocations and needs to only be concerned with domain specific variables such as ”atoms”, ”elements”, ”time steps” or ”iterations”. The scheduler is able to then chose the best resource allocation to get the fastest turn-around-time. If the system is very busy then the scheduler may make a sub-optimal allocation as long as the execution times and associated wait times

are lower for the sub-optimal. While still susceptible to bad-put the system can never the less ensure maximum utilisation by moulding jobs to fit gaps. The current implementation is computationally expensive but in a First Come First Served (FCFS) environment these are insignificant costs.

To tackle the challenge of computational complexity and expense an inference decision engine based on the Mamdani algorithm for Fuzzy Logic has been presented. This system is able to reduce the number of states during which the scheduler attempts to mould a job, reducing the computation by 40%. This method once implemented can help the mouldable scheduler to be adapted to attempt out-of-order scheduling. Mouldable scheduling in a first come first served with backfilling environment, coupled with very close predictions of expected job run times will solve the problem of efficiency vs. Quality of Service (QoS). QoS is guaranteed as no out-of-order execution will take place if it delays the top of the queue. But with moulding inevitably some job in the queue will fit into idle resources, maximising utilisation.

A method of surge computing, which has been deployed in a production environment is also presented in the thesis as the next stage of advanced scheduling. The surge computing add-on is not only able to scale beyond institutional firewalls to deal with high load but it is also able to provision for resource allocations that are not offered or available on the underlying HPC system.

To test the systems, methods and algorithms presented in this thesis a simulator was also designed and implemented. The Cluster Discrete Event Simulator (CDES) is a highly flexible scheduler which can simulate with near perfect accuracy a real HPC system. With no user or fair-share intervention the simulator matched the scheduling decisions of the real cluster 100% of the time. The design and results of this simulator has been peer-reviewed and published on two separate occasions.

Alongside the simulator a ”real” workload from a production HPC at the University of Huddersfield was utilised. Since there are no standard mouldable workloads, as no mouldable schedulers exist in production, the real workload had to be augmented with application information. These augmented trace logs were further synthesised using standards outlined by other pioneers in the field of mouldable scheduling. To perform more accurate testing the mouldable scheduler has to be moved from a simulated world to a real world system. This is because in a simulated environment a job ends at the expected calculated time, while in a real world system a job may complete earlier or later that the calculated end time.

Using logs from a busy month on the Eridani HPC system and tracking four users in particular, the mouldable scheduler was able to dramatically reduce the overall system turn-around-time. Reducing a workload that took 38 days for processing to 30 days is a significant improvement. Scaled to large systems cost savings are unparalleled. An 8 day improvement in job turn-around-time for the end users is also very important. Such an improvement would be remarkable for QoS guarantees. Another significant result was that the scheduler did not improve turn-around-time by shrinking allocations. In fact for a majority of the moulded jobs the scheduler in the first instance provisioned more resources per job. After initially moulding the jobs to ”optimum” allocations, the system proceeded to sub-optimally mould jobs just about 5% of the time. These results also highlighted that contrary to popular assumption, bad-put can also exist in the form of under-booking resources. The belief that choosing the minimum possible allocation to get maximum throughput may not always produce the fastest turn-around-time for the whole workload.

With applications being delivered to end-users through clouds and users migrating their workloads into the cloud, the new dynamic and elastic nature of computing is here to stay. For traditionally compute intensive workloads to adapt to

this new paradigms an Intelligent Robust Mouldable Scheduler is required.

Chapter 11

Further Work

To reach the target of designing a mouldable scheduler using real application benchmarks several key milestones needed to be achieved. While not mature in their own right these milestones (e.g. the creation of an application profiling suite or a High Performance Computing (HPC) workload simulator) were sufficient for the purpose of this project. Each of the Chapters from 6 to 9 are all work in progress and there is still much research and much refinement to be done. Most of the products created are of Technology Readiness Level (TRL) 4-6. This chapter briefly outlines some of the on-going research and improvements that can be undertaken.

To make the Application and System Performance Profiler (ASPP) a robust and stand alone tool for use in industry code hardening is required. The profilers also needs to incorporate memory and network activity. This information will help greatly in purchase time decisions and can aid the mouldable scheduler to better predict the upper and lower bounds of resource allocations. The evaluator module from the mouldable scheduler can be incorporated into the benchmarking tool to provide end users a new dimension to their benchmarking.

In the context of the mouldable scheduler, the linearisation and weighted averages technique utilised within the evaluator was a highly effective way to quickly and computationally inexpensive get results to benchmark against. Utilising curve fitting or ideally surface modelling algorithms should improve the predicted values. Curve fitting will automatically correct the problem faced when extrapolating lines. Curves will be able to give the system an indication of when diminishing returns will be observed.

Further the out-of-order execution algorithms as used to test Cluster Discrete Event Simulator (CDES) should be used to assess effectiveness of the mouldable scheduler. Standard job logs do not contain enough information for such a test so either logs need to be synthesised or new datasets with the information augmented need to be created from real workloads.

The CDES system has been used in other research projects to validate Brennan's results (J. D. Brennan, 2014) for scaling campus grids. As outlined in Section 7.5 the existing algorithms are no able to cater to user interventions in the trace log (i.e. a user kills a job that has not yet run). This and similar bugs need to be corrected before wider dissemination is possible.

To allow other system administrators and developers to utilise the CDES system, the API needs to be documented and base functions within the code need to be made flexible. Others using the CDES will not need to replicate certain functionality within their algorithms. For wider usability CDES needs to include more algorithms by default for easier uptake.

While the mouldable scheduler has shown considerable improvement in job turn-around-times under simulator settings (TRL5), it requires testing in a real world deployment (TRL6/7). As discussed in Section 8.3 predicted end times used in scheduling are not fixed times. A job may exceed the prediction or end before (usually because of application behaviour). A simulated environment

can not recreate this behaviour effectively and thus any testing carried out for mouldable or even malleable scheduling can only rely on normalised data.

With real world deployment the Surge Computing add-ons discussed in Section 9.3 can benefit with the further layers of dynamism. As shown in Section 9.3.2 surging still relies on a user to prescribe their memory and processor requirements before a virtual machine is spawned. With many research groups moving towards cloud computing solutions for their HPC needs, an automated surging facility which can decide resource allocations based on real application benchmarks will make cloud migration easier. This will also open a new variable for the mouldable scheduler to base decisions on, that is price.

As more complex workloads and system configurations are implemented and tested the computational overheads will increase. To aide in improving the performance of the simulator (and later a production mouldable scheduler) the fuzzy logic algorithm discussed in 9.2 needs to be implemented. Work is underway to use the py_fuzzy toolbox available for Python 1.7 to implement the membership functions for integration with CDES.

In literature available for mouldable or malleable scheduling, no realistic method is ever presented to help the system determine expected end times for different resource allocations. The ASPP solution gives the scheduler a real chance to predict end times. However, benchmarks are general purpose and it is too expensive for every application to be benchmarked across the system. The work in this thesis relies on system administrators benchmarking their system at time of delivery as part of compliance testing. But new applications, new projects - with new datasets and workloads, and new users can be introduced to the system over time. Benchmarking for every situation is impossible. Further the mouldable scheduler outlined here continues to rely on some user input and

does not incorporate any adaptation process to correct for the inevitable bad put where a user does not change their input file.

To counteract these elements machine and deep learning strategies have been identified to help profile user activities and create models so that the scheduler can adapt. Modelling the user with heuristic information will allow the system to identify if a job is an outlier - possibly due to bad put, or can identify the exact requirements of the user which may be different from the general benchmarks.

Appendix A

Appendix A: Dataset

This appendix contains the trace logs for the four users that made up the parallel workloads testing on this system. These include the two Computational Fluid Dynamics (CFD) users and the two Molecular Dynamics (MD) users The logs a truncated to save space. 10 records per user are presented in the following sections. The full logs may be downloaded from:
http://www.ibadkureshi.com/thesis/appendix-a.zip.

A.1 Real Tracelogs

```
28/04/2013 10:21:15;E;73207.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366897046 qtime=1366897046 etime=1366897046 start=1366970945 end=1367140875
    Resource_List.nodes=2:ppn=4 resources_used.walltime=47:12:10
27/04/2013 13:24:58;E;73269.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366898750 qtime=1366898750 etime=1366898750 start=1366970967 end=1367065498
    Resource_List.nodes=2:ppn=4 resources_used.walltime=26:15:31
27/04/2013 11:42:08;E;73002.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366889417 qtime=1366889417 etime=1366889417 start=1366889417 end=1367059328
    Resource_List.nodes=2:ppn=4 resources_used.walltime=47:11:51
27/04/2013 04:50:50;E;75252.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982459 qtime=1366982459 etime=1366982459 start=1367034615 end=1367034650
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:35
```

```
27/04/2013 04:49:56;E;75251.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982458 qtime=1366982458 etime=1366982458 start=1367034545 end=1367034596
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:51
27/04/2013 04:48:58;E;75250.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1366982457 etime=1366982457 start=1367034514 end=1367034538
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:24
27/04/2013 04:48:05;E;75249.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1366982457 etime=1366982457 start=1367034457 end=1367034485
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:28
27/04/2013 04:47:26;E;75248.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1366982457 etime=1366982457 start=1367034428 end=1367034446
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:18
27/04/2013 04:46:57;E;75247.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982456 qtime=1366982456 etime=1366982456 start=1367034403 end=1367034417
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:14
27/04/2013 04:46:30;E;75246.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982454 qtime=1366982454 etime=1366982454 start=1367034373 end=1367034390
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:17
04/03/2013 15:31:12;E;77747.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999418 qtime=1364999418 etime=1364999418 start
    =1364999418 owner=mduser2@qgg.hud.ac.uk exec_host=enode32.eridani.qgg.hud.ac.uk/3+
    enode32.eridani.qgg.hud.ac.uk/2+enode32.eridani.qgg.hud.ac.uk/1+enode32.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=4398
    end=1364999472 Exit_status=271 resources_used.cput=00:03:18 resources_used.mem
    =37392kb resources_used.vmem=969072kb resources_used.walltime=00:00:54
04/03/2013 22:10:07;E;77761.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999649 qtime=1364999649 etime=1364999649 start
    =1364999649 owner=mduser2@qgg.hud.ac.uk exec_host=enode35.eridani.qgg.hud.ac.uk/3+
    enode35.eridani.qgg.hud.ac.uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=7680
    end=1365023407 Exit_status=0 resources_used.cput=26:22:14 resources_used.mem=37372
    kb resources_used.vmem=967404kb resources_used.walltime=06:35:58
04/04/2013 01:13:50;E;77778.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1365000012 qtime=1365000012 etime=1365000012 start
    =1365018811 owner=mduser2@qgg.hud.ac.uk exec_host=enode19.eridani.qgg.hud.ac.uk/3+
    enode19.eridani.qgg.hud.ac.uk/2+enode19.eridani.qgg.hud.ac.uk/1+enode19.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=21974
     end=1365034430 Exit_status=0 resources_used.cput=17:20:07 resources_used.mem
    =37372kb resources_used.vmem=968520kb resources_used.walltime=04:20:20
```

```
04/04/2013 02:23:07;E;77776.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999966 qtime=1364999966 etime=1364999966 start
    =1365018810 owner=mduser2@qgg.hud.ac.uk exec_host=enode18.eridani.qgg.hud.ac.uk/3+
    enode18.eridani.qgg.hud.ac.uk/2+enode18.eridani.qgg.hud.ac.uk/1+enode18.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=21176
     end=1365038587 Exit_status=0 resources_used.cput=21:57:14 resources_used.mem
    =37388kb resources_used.vmem=968520kb resources_used.walltime=05:29:37
04/04/2013 02:49:41;E;77775.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999921 qtime=1364999921 etime=1364999921 start
    =1365017691 owner=mduser2@qgg.hud.ac.uk exec_host=enode23.eridani.qgg.hud.ac.uk/3+
    enode23.eridani.qgg.hud.ac.uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=12655
     end=1365040181 Exit_status=0 resources_used.cput=24:57:36 resources_used.mem
    =37532kb resources_used.vmem=968528kb resources_used.walltime=06:14:50
04/04/2013 09:07:40;E;77738.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999320 qtime=1364999320 etime=1364999320 start
    =1364999320 owner=mduser2@qgg.hud.ac.uk exec_host=enode30.eridani.qgg.hud.ac.uk/3+
    enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=7671
    end=1365062860 Exit_status=0 resources_used.cput=70:31:43 resources_used.mem=36960
    kb resources_used.vmem=966112kb resources_used.walltime=17:39:00
04/04/2013 12:07:07;E;77768.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999749 qtime=1364999749 etime=1364999749 start
    =1365002242 owner=mduser2@qgg.hud.ac.uk exec_host=enode34.eridani.qgg.hud.ac.uk/3+
    enode34.eridani.qgg.hud.ac.uk/2+enode34.eridani.qgg.hud.ac.uk/1+enode34.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=8017
    end=1365073627 Exit_status=0 resources_used.cput=79:14:18 resources_used.mem=37208
    kb resources_used.vmem=967288kb resources_used.walltime=19:49:45
04/04/2013 12:28:22;E;77733.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999241 qtime=1364999241 etime=1364999241 start
    =1364999241 owner=mduser2@qgg.hud.ac.uk exec_host=enode28.eridani.qgg.hud.ac.uk/3+
    enode28.eridani.qgg.hud.ac.uk/2+enode28.eridani.qgg.hud.ac.uk/1+enode28.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=8773
    end=1365074902 Exit_status=0 resources_used.cput=83:58:44 resources_used.mem=37272
    kb resources_used.vmem=966120kb resources_used.walltime=21:01:01
```

```
04/23/2013 09:18:53;E;101863.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=paraul ctime=1366640324 qtime=1366640324 etime=1366640324 start
    =1366640325 owner=mduser2@qgg.hud.ac.uk exec_host=enode35.eridani.qgg.hud.ac.uk/3+
    enode35.eridani.qgg.hud.ac.uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.eridani.
    qgg.hud.ac.uk/0+enode34.eridani.qgg.hud.ac.uk/3+enode34.eridani.qgg.hud.ac.uk/2+
    enode34.eridani.qgg.hud.ac.uk/1+enode34.eridani.qgg.hud.ac.uk/0+enode33.eridani.
    qgg.hud.ac.uk/3+enode33.eridani.qgg.hud.ac.uk/2+enode33.eridani.qgg.hud.ac.uk/1+
    enode33.eridani.qgg.hud.ac.uk/0+enode32.eridani.qgg.hud.ac.uk/3+enode32.eridani.
    qgg.hud.ac.uk/2+enode32.eridani.qgg.hud.ac.uk/1+enode32.eridani.qgg.hud.ac.uk/0+
    enode31.eridani.qgg.hud.ac.uk/3+enode31.eridani.qgg.hud.ac.uk/2+enode31.eridani.
    qgg.hud.ac.uk/1+enode31.eridani.qgg.hud.ac.uk/0+enode30.eridani.qgg.hud.ac.uk/3+
    enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.eridani.
    qgg.hud.ac.uk/0+enode29.eridani.qgg.hud.ac.uk/3+enode29.eridani.qgg.hud.ac.uk/2+
    enode29.eridani.qgg.hud.ac.uk/1+enode29.eridani.qgg.hud.ac.uk/0+enode28.eridani.
    qgg.hud.ac.uk/3+enode28.eridani.qgg.hud.ac.uk/2+enode28.eridani.qgg.hud.ac.uk/1+
    enode28.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List.ncpus
    =1 Resource_List.neednodes=8:ppn=4 Resource_List.nodect=8 Resource_List.nodes=8:
    ppn=4 Resource_List.walltime=336:00:00 session=29825 end=1366705133 Exit_status=0
    resources_used.cput=11:22:01 resources_used.mem=5823168kb resources_used.vmem
    =39016268kb resources_used.walltime=18:00:08
04/23/2013 09:43:57;E;101864.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=paraul ctime=1366706619 qtime=1366706619 etime=1366706619 start
    =1366706619 owner=mduser2@qgg.hud.ac.uk exec_host=enode35.eridani.qgg.hud.ac.uk/3+
    enode35.eridani.qgg.hud.ac.uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.eridani.
    qgg.hud.ac.uk/0+enode34.eridani.qgg.hud.ac.uk/3+enode34.eridani.qgg.hud.ac.uk/2+
    enode34.eridani.qgg.hud.ac.uk/1+enode34.eridani.qgg.hud.ac.uk/0+enode33.eridani.
    qgg.hud.ac.uk/3+enode33.eridani.qgg.hud.ac.uk/2+enode33.eridani.qgg.hud.ac.uk/1+
    enode33.eridani.qgg.hud.ac.uk/0+enode32.eridani.qgg.hud.ac.uk/3+enode32.eridani.
    qgg.hud.ac.uk/2+enode32.eridani.qgg.hud.ac.uk/1+enode32.eridani.qgg.hud.ac.uk/0+
    enode31.eridani.qgg.hud.ac.uk/3+enode31.eridani.qgg.hud.ac.uk/2+enode31.eridani.
    qgg.hud.ac.uk/1+enode31.eridani.qgg.hud.ac.uk/0+enode30.eridani.qgg.hud.ac.uk/3+
    enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.eridani.
    qgg.hud.ac.uk/0+enode29.eridani.qgg.hud.ac.uk/3+enode29.eridani.qgg.hud.ac.uk/2+
    enode29.eridani.qgg.hud.ac.uk/1+enode29.eridani.qgg.hud.ac.uk/0+enode28.eridani.
    qgg.hud.ac.uk/3+enode28.eridani.qgg.hud.ac.uk/2+enode28.eridani.qgg.hud.ac.uk/1+
    enode28.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List.ncpus
    =1 Resource_List.neednodes=8:ppn=4 Resource_List.nodect=8 Resource_List.nodes=8:
    ppn=4 Resource_List.walltime=336:00:00 session=30192 end=1366706637 Exit_status=0
    resources_used.cput=00:00:00 resources_used.mem=824kb resources_used.vmem=13380kb
    resources_used.walltime=00:00:15
```

```
04/02/2013 00:13:54;E;77182.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case1 queue=paraul ctime=1364858024 qtime=1364858024 etime=1364858024 start
    =1364858024 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode19.eridani.qgg.hud
    .ac.uk/3+enode19.eridani.qgg.hud.ac.uk/2+enode19.eridani.qgg.hud.ac.uk/1+enode19.
    eridani.qgg.hud.ac.uk/0+enode18.eridani.qgg.hud.ac.uk/3+enode18.eridani.qgg.hud.ac
    .uk/2+enode18.eridani.qgg.hud.ac.uk/1+enode18.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
    =336:00:00 session=18948 end=1364858034 Exit_status=271 resources_used.cput
    =00:00:00 resources_used.mem=41296kb resources_used.vmem=117408kb resources_used.
    walltime=00:00:10
04/02/2013 00:14:03;E;77181.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case237 queue=paraul ctime=1364858011 qtime=1364858011 etime=1364858011 start
    =1364858011 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode17.eridani.qgg.hud
    .ac.uk/3+enode17.eridani.qgg.hud.ac.uk/2+enode17.eridani.qgg.hud.ac.uk/1+enode17.
    eridani.qgg.hud.ac.uk/0+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
    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
    =336:00:00 session=20312 end=1364858043 Exit_status=0 resources_used.cput=00:00:00
     resources_used.mem=112764kb resources_used.vmem=1343808kb resources_used.walltime
    =00:00:32
04/02/2013 00:16:22;E;77183.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case238 queue=paraul ctime=1364858154 qtime=1364858154 etime=1364858154 start
    =1364858154 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode17.eridani.qgg.hud
    .ac.uk/3+enode17.eridani.qgg.hud.ac.uk/2+enode17.eridani.qgg.hud.ac.uk/1+enode17.
    eridani.qgg.hud.ac.uk/0+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
    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
    =336:00:00 session=21285 end=1364858182 Exit_status=0 resources_used.cput=00:00:00
     resources_used.mem=88076kb resources_used.vmem=1068672kb resources_used.walltime
    =00:00:28
```

```
04/02/2013 00:16:31;E;77184.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case239 queue=paraul ctime=1364858161 qtime=1364858161 etime=1364858161 start
    =1364858161 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode19.eridani.qgg.hud
    .ac.uk/3+enode19.eridani.qgg.hud.ac.uk/2+enode19.eridani.qgg.hud.ac.uk/1+enode19.
    eridani.qgg.hud.ac.uk/0+enode18.eridani.qgg.hud.ac.uk/3+enode18.eridani.qgg.hud.ac
    .uk/2+enode18.eridani.qgg.hud.ac.uk/1+enode18.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
    =336:00:00 session=19641 end=1364858191 Exit_status=0 resources_used.cput=00:00:00
     resources_used.mem=65360kb resources_used.vmem=658444kb resources_used.walltime
    =00:00:30
04/02/2013 00:16:34;E;77185.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case240 queue=paraul ctime=1364858167 qtime=1364858167 etime=1364858167 start
    =1364858167 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode22.eridani.qgg.hud
    .ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.
    eridani.qgg.hud.ac.uk/0+enode21.eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac
    .uk/2+enode21.eridani.qgg.hud.ac.uk/1+enode21.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
    =336:00:00 session=21144 end=1364858194 Exit_status=0 resources_used.cput=00:00:00
     resources_used.mem=3004kb resources_used.vmem=95216kb resources_used.walltime
    =00:00:27
04/02/2013 00:16:36;E;77186.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case241 queue=paraul ctime=1364858172 qtime=1364858172 etime=1364858172 start
    =1364858172 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode24.eridani.qgg.hud
    .ac.uk/3+enode24.eridani.qgg.hud.ac.uk/2+enode24.eridani.qgg.hud.ac.uk/1+enode24.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.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
    =336:00:00 session=13325 end=1364858196 Exit_status=0 resources_used.cput=00:00:00
     resources_used.mem=816kb resources_used.vmem=13380kb resources_used.walltime
    =00:00:24
```

```
04/02/2013 00:16:55;E;77187.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case242 queue=paraul ctime=1364858182 qtime=1364858182 etime=1364858182 start
    =1364858182 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode26.eridani.qgg.hud
    .ac.uk/3+enode26.eridani.qgg.hud.ac.uk/2+enode26.eridani.qgg.hud.ac.uk/1+enode26.
    eridani.qgg.hud.ac.uk/0+enode25.eridani.qgg.hud.ac.uk/3+enode25.eridani.qgg.hud.ac
    .uk/2+enode25.eridani.qgg.hud.ac.uk/1+enode25.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
    =336:00:00 session=9402 end=1364858215 Exit_status=0 resources_used.cput=00:00:00
    resources_used.mem=60192kb resources_used.vmem=529896kb resources_used.walltime
    =00:00:33
04/02/2013 13:58:02;E;77192.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case239 queue=paraul ctime=1364907453 qtime=1364907453 etime=1364907453 start
    =1364907453 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode22.eridani.qgg.hud
    .ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.
    eridani.qgg.hud.ac.uk/0+enode21.eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac
    .uk/2+enode21.eridani.qgg.hud.ac.uk/1+enode21.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
    =336:00:00 session=22282 end=1364907482 Exit_status=0 resources_used.cput=00:00:00
     resources_used.mem=816kb resources_used.vmem=13380kb resources_used.walltime
    =00:00:29
04/02/2013 13:58:15;E;77193.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case240 queue=paraul ctime=1364907469 qtime=1364907469 etime=1364907469 start
    =1364907469 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode24.eridani.qgg.hud
    .ac.uk/3+enode24.eridani.qgg.hud.ac.uk/2+enode24.eridani.qgg.hud.ac.uk/1+enode24.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.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
    =336:00:00 session=14463 end=1364907495 Exit_status=0 resources_used.cput=00:00:00
     resources_used.mem=69828kb resources_used.vmem=763400kb resources_used.walltime
    =00:00:26
```

```
04/02/2013 13:58:24;E;77194.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case241 queue=paraul ctime=1364907481 qtime=1364907481 etime=1364907481 start
    =1364907481 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode26.eridani.qgg.hud
    .ac.uk/3+enode26.eridani.qgg.hud.ac.uk/2+enode26.eridani.qgg.hud.ac.uk/1+enode26.
    eridani.qgg.hud.ac.uk/0+enode25.eridani.qgg.hud.ac.uk/3+enode25.eridani.qgg.hud.ac
    .uk/2+enode25.eridani.qgg.hud.ac.uk/1+enode25.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
    =336:00:00 session=10540 end=1364907504 Exit_status=0 resources_used.cput=00:00:00
     resources_used.mem=65084kb resources_used.vmem=659600kb resources_used.walltime
    =00:00:23
04/04/2013 01:51:38;E;78715.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365027644 qtime=1365027644 etime=1365027644 start
    =1365036667 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode35.eridani.qgg.hud
    .ac.uk/3+enode35.eridani.qgg.hud.ac.uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.
    eridani.qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac
    .uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.eridani.qgg.hud.ac.uk/0+enode21.
    eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac.uk/2+enode21.eridani.qgg.hud.ac
    .uk/1+enode21.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 Resource_List.walltime=336:00:00 session=23365 end=1365036698
    Exit_status=0 resources_used.cput=00:00:01 resources_used.mem=64984kb
    resources_used.vmem=787496kb resources_used.walltime=00:00:31
04/04/2013 11:27:50;E;79221.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365071243 qtime=1365071243 etime=1365071243 start
    =1365071243 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode23.eridani.qgg.hud
    .ac.uk/3+enode23.eridani.qgg.hud.ac.uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.
    eridani.qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac
    .uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.eridani.qgg.hud.ac.uk/0+enode21.
    eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac.uk/2+enode21.eridani.qgg.hud.ac
    .uk/1+enode21.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 Resource_List.walltime=336:00:00 session=14219 end=1365071270
    Exit_status=0 resources_used.cput=00:00:01 resources_used.mem=6868kb
    resources_used.vmem=202120kb resources_used.walltime=00:00:27
```

```
04/04/2013 11:41:35;E;79251.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365072064 qtime=1365072064 etime=1365072064 start
    =1365072064 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode30.eridani.qgg.hud
    .ac.uk/3+enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.eridani.qgg.hud.ac.uk/0+enode22.
    eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac
    .uk/1+enode22.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 Resource_List.walltime=336:00:00 session=8494 end=1365072095
    Exit_status=0 resources_used.cput=00:00:06 resources_used.mem=4596kb
    resources_used.vmem=142224kb resources_used.walltime=00:00:31
04/04/2013 12:18:43;E;79348.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365074292 qtime=1365074292 etime=1365074292 start
    =1365074292 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode30.eridani.qgg.hud
    .ac.uk/3+enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.eridani.qgg.hud.ac.uk/0+enode22.
    eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac
    .uk/1+enode22.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 Resource_List.walltime=336:00:00 session=9544 end=1365074323
    Exit_status=0 resources_used.cput=00:00:00 resources_used.mem=65684kb
    resources_used.vmem=851132kb resources_used.walltime=00:00:31
04/04/2013 12:25:13;E;79365.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365074685 qtime=1365074685 etime=1365074685 start
    =1365074686 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode30.eridani.qgg.hud
    .ac.uk/3+enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.eridani.qgg.hud.ac.uk/0+enode22.
    eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac
    .uk/1+enode22.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 Resource_List.walltime=336:00:00 session=10584 end=1365074713
    Exit_status=0 resources_used.cput=00:00:00 resources_used.mem=90080kb
    resources_used.vmem=1330968kb resources_used.walltime=00:00:27
```

```
04/04/2013 12:37:55;E;79397.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365075452 qtime=1365075452 etime=1365075452 start
    =1365075452 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode23.eridani.qgg.hud
    .ac.uk/3+enode23.eridani.qgg.hud.ac.uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.
    eridani.qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac
    .uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.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
    =336:00:00 session=15483 end=1365075475 Exit_status=0 resources_used.cput=00:00:01
     resources_used.mem=120120kb resources_used.vmem=1384756kb resources_used.walltime
    =00:00:23
04/04/2013 12:44:24;E;79409.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365075736 qtime=1365075736 etime=1365075736 start
    =1365075736 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode23.eridani.qgg.hud
    .ac.uk/3+enode23.eridani.qgg.hud.ac.uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.
    eridani.qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac
    .uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.eridani.qgg.hud.ac.uk/0+enode21.
    eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac.uk/2+enode21.eridani.qgg.hud.ac
    .uk/1+enode21.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 Resource_List.walltime=336:00:00 session=16452 end=1365075864
    Exit_status=0 resources_used.cput=00:06:08 resources_used.mem=2359792kb
    resources_used.vmem=4252684kb resources_used.walltime=00:02:08
04/04/2013 12:53:33;E;79432.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365076375 qtime=1365076375 etime=1365076375 start
    =1365076375 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode37.eridani.qgg.hud
    .ac.uk/3+enode37.eridani.qgg.hud.ac.uk/2+enode37.eridani.qgg.hud.ac.uk/1+enode37.
    eridani.qgg.hud.ac.uk/0+enode35.eridani.qgg.hud.ac.uk/3+enode35.eridani.qgg.hud.ac
    .uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.eridani.qgg.hud.ac.uk/0+enode34.
    eridani.qgg.hud.ac.uk/3+enode34.eridani.qgg.hud.ac.uk/2+enode34.eridani.qgg.hud.ac
    .uk/1+enode34.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 Resource_List.walltime=336:00:00 session=7069 end=1365076413
    Exit_status=0 resources_used.cput=00:00:00 resources_used.mem=816kb resources_used
    .vmem=13380kb resources_used.walltime=00:00:38
```

```
04/04/2013 13:02:53;E;79462.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365076875 qtime=1365076876 etime=1365076876 start
    =1365076876 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode34.eridani.qgg.hud
    .ac.uk/3+enode34.eridani.qgg.hud.ac.uk/2+enode34.eridani.qgg.hud.ac.uk/1+enode34.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.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
    =336:00:00 session=8921 end=1365076973 Exit_status=0 resources_used.cput=00:04:16
    resources_used.mem=1953136kb resources_used.vmem=2599216kb resources_used.walltime
    =00:01:37
04/04/2013 13:34:52;E;79537.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365078333 qtime=1365078333 etime=1365078333 start
    =1365078703 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode37.eridani.qgg.hud
    .ac.uk/3+enode37.eridani.qgg.hud.ac.uk/2+enode37.eridani.qgg.hud.ac.uk/1+enode37.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.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
    =336:00:00 session=8693 end=1365078892 Exit_status=0 resources_used.cput=00:10:39
    resources_used.mem=3347388kb resources_used.vmem=4628624kb resources_used.walltime
    =00:03:09
```

A.2 Real Data with Moulding Information

```
27/04/2013 22:54:03;E;73207.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366897046 qtime=1366897046 etime=1366897046 start
    =1366970945 owner=mduser1@qgg.hud.ac.uk exec_host=enode17.eridani.qgg.hud.ac.uk/3+
    enode17.eridani.qgg.hud.ac.uk/2+enode17.eridani.qgg.hud.ac.uk/1+enode17.eridani.
    qgg.hud.ac.uk/0+enode15.eridani.qgg.hud.ac.uk/3+enode15.eridani.qgg.hud.ac.uk/2+
    enode15.eridani.qgg.hud.ac.uk/1+enode15.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 session=816 end=1367099643 Exit_status=0
    resources_used.cput=188:42:46 resources_used.walltime=35:44:58 app_name=dlpoly
    iterations=900000 atoms=4000
```

```
27/04/2013 22:54:25;E;73269.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366898750 qtime=1366898750 etime=1366898750 start
    =1366970967 owner=mduser1@qgg.hud.ac.uk exec_host=enode25.eridani.qgg.hud.ac.uk/3+
    enode25.eridani.qgg.hud.ac.uk/2+enode25.eridani.qgg.hud.ac.uk/1+enode25.eridani.
    qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+
    enode22.eridani.qgg.hud.ac.uk/1+enode22.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 session=19341 end=1367099665 Exit_status=0
    resources_used.cput=104:57:17 resources_used.walltime=35:44:58 app_name=dlpoly
    iterations=900000 atoms=4000
27/04/2013 00:15:15;E;73002.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366889417 qtime=1366889417 etime=1366889417 start
    =1366889417 owner=mduser1@qgg.hud.ac.uk exec_host=enode05.eridani.qgg.hud.ac.uk/3+
    enode05.eridani.qgg.hud.ac.uk/2+enode05.eridani.qgg.hud.ac.uk/1+enode05.eridani.
    qgg.hud.ac.uk/0+enode04.eridani.qgg.hud.ac.uk/3+enode04.eridani.qgg.hud.ac.uk/2+
    enode04.eridani.qgg.hud.ac.uk/1+enode04.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 session=2283 end=1367018115 Exit_status=0
    resources_used.cput=188:43:16 resources_used.walltime=35:44:58 app_name=dlpoly
    iterations=900000 atoms=4000
27/04/2013 04:50:58;E;75252.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366982459 qtime=1366982459 etime=1366982459 start
    =1367034615 owner=mduser1@qgg.hud.ac.uk exec_host=enode09.eridani.qgg.hud.ac.uk/3+
    enode09.eridani.qgg.hud.ac.uk/2+enode09.eridani.qgg.hud.ac.uk/1+enode09.eridani.
    qgg.hud.ac.uk/0+enode01.eridani.qgg.hud.ac.uk/3+enode01.eridani.qgg.hud.ac.uk/2+
    enode01.eridani.qgg.hud.ac.uk/1+enode01.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 session=14801 end=1367034658 Exit_status=0
    resources_used.cput=00:00:00 resources_used.walltime=00:00:43 app_name=dlpoly
    iterations=300 atoms=4000
27/04/2013 04:49:48;E;75251.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366982458 qtime=1366982458 etime=1366982458 start
    =1367034545 owner=mduser1@qgg.hud.ac.uk exec_host=enode09.eridani.qgg.hud.ac.uk/3+
    enode09.eridani.qgg.hud.ac.uk/2+enode09.eridani.qgg.hud.ac.uk/1+enode09.eridani.
    qgg.hud.ac.uk/0+enode01.eridani.qgg.hud.ac.uk/3+enode01.eridani.qgg.hud.ac.uk/2+
    enode01.eridani.qgg.hud.ac.uk/1+enode01.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 session=14725 end=1367034588 Exit_status=0
    resources_used.cput=00:00:00 resources_used.walltime=00:00:43 app_name=dlpoly
    iterations=300 atoms=4000
```

```
27/04/2013 04:49:17;E;75250.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366982457 qtime=1366982457 etime=1366982457 start
    =1367034514 owner=mduser1@qgg.hud.ac.uk exec_host=enode09.eridani.qgg.hud.ac.uk/3+
    enode09.eridani.qgg.hud.ac.uk/2+enode09.eridani.qgg.hud.ac.uk/1+enode09.eridani.
    qgg.hud.ac.uk/0+enode01.eridani.qgg.hud.ac.uk/3+enode01.eridani.qgg.hud.ac.uk/2+
    enode01.eridani.qgg.hud.ac.uk/1+enode01.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 session=14649 end=1367034557 Exit_status=0
    resources_used.cput=00:00:00 resources_used.walltime=00:00:43 app_name=dlpoly
    iterations=300 atoms=4000
27/04/2013 04:48:20;E;75249.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366982457 qtime=1366982457 etime=1366982457 start
    =1367034457 owner=mduser1@qgg.hud.ac.uk exec_host=enode09.eridani.qgg.hud.ac.uk/3+
    enode09.eridani.qgg.hud.ac.uk/2+enode09.eridani.qgg.hud.ac.uk/1+enode09.eridani.
    qgg.hud.ac.uk/0+enode01.eridani.qgg.hud.ac.uk/3+enode01.eridani.qgg.hud.ac.uk/2+
    enode01.eridani.qgg.hud.ac.uk/1+enode01.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 session=14573 end=1367034500 Exit_status=0
    resources_used.cput=00:00:00 resources_used.walltime=00:00:43 app_name=dlpoly
    iterations=300 atoms=4000
27/04/2013 04:47:51;E;75248.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366982457 qtime=1366982457 etime=1366982457 start
    =1367034428 owner=mduser1@qgg.hud.ac.uk exec_host=enode09.eridani.qgg.hud.ac.uk/3+
    enode09.eridani.qgg.hud.ac.uk/2+enode09.eridani.qgg.hud.ac.uk/1+enode09.eridani.
    qgg.hud.ac.uk/0+enode01.eridani.qgg.hud.ac.uk/3+enode01.eridani.qgg.hud.ac.uk/2+
    enode01.eridani.qgg.hud.ac.uk/1+enode01.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 session=14497 end=1367034471 Exit_status=0
    resources_used.cput=00:00:00 resources_used.walltime=00:00:43 app_name=dlpoly
    iterations=300 atoms=4000
27/04/2013 04:47:26;E;75247.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366982456 qtime=1366982456 etime=1366982456 start
    =1367034403 owner=mduser1@qgg.hud.ac.uk exec_host=enode09.eridani.qgg.hud.ac.uk/3+
    enode09.eridani.qgg.hud.ac.uk/2+enode09.eridani.qgg.hud.ac.uk/1+enode09.eridani.
    qgg.hud.ac.uk/0+enode01.eridani.qgg.hud.ac.uk/3+enode01.eridani.qgg.hud.ac.uk/2+
    enode01.eridani.qgg.hud.ac.uk/1+enode01.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 session=14421 end=1367034446 Exit_status=0
    resources_used.cput=00:00:00 resources_used.walltime=00:00:43 app_name=dlpoly
    iterations=300 atoms=4000
```

```
27/04/2013 04:46:56;E;75246.eridani.qgg.hud.ac.uk;user=mduser1 group=pgr jobname=
    dlpoly queue=parastd ctime=1366982454 qtime=1366982454 etime=1366982454 start
    =1367034373 owner=mduser1@qgg.hud.ac.uk exec_host=enode09.eridani.qgg.hud.ac.uk/3+
    enode09.eridani.qgg.hud.ac.uk/2+enode09.eridani.qgg.hud.ac.uk/1+enode09.eridani.
    qgg.hud.ac.uk/0+enode01.eridani.qgg.hud.ac.uk/3+enode01.eridani.qgg.hud.ac.uk/2+
    enode01.eridani.qgg.hud.ac.uk/1+enode01.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 session=14345 end=1367034416 Exit_status=0
    resources_used.cput=00:00:00 resources_used.walltime=00:00:43 app_name=dlpoly
    iterations=300 atoms=4000
04/04/2013 14:31:08;E;77747.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999418 qtime=1364999418 etime=1364999418 start
    =1364999418 owner=mduser2@qgg.hud.ac.uk exec_host=enode32.eridani.qgg.hud.ac.uk/3+
    enode32.eridani.qgg.hud.ac.uk/2+enode32.eridani.qgg.hud.ac.uk/1+enode32.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=4398
    end=1365082268 Exit_status=271 resources_used.cput=00:03:18 resources_used.
    walltime=23:00:50 app_name=dlpoly iterations=5000000 atoms=500
04/04/2013 14:34:59;E;77761.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999649 qtime=1364999649 etime=1364999649 start
    =1364999649 owner=mduser2@qgg.hud.ac.uk exec_host=enode35.eridani.qgg.hud.ac.uk/3+
    enode35.eridani.qgg.hud.ac.uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=7680
    end=1365082499 Exit_status=0 resources_used.cput=26:22:14 resources_used.walltime
    =23:00:50 app_name=dlpoly iterations=5000000 atoms=500
04/04/2013 19:54:21;E;77778.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1365000012 qtime=1365000012 etime=1365000012 start
    =1365018811 owner=mduser2@qgg.hud.ac.uk exec_host=enode19.eridani.qgg.hud.ac.uk/3+
    enode19.eridani.qgg.hud.ac.uk/2+enode19.eridani.qgg.hud.ac.uk/1+enode19.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=21974
     end=1365101661 Exit_status=0 resources_used.cput=17:20:07 resources_used.walltime
    =23:00:50 app_name=dlpoly iterations=5000000 atoms=500
04/04/2013 19:54:20;E;77776.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999966 qtime=1364999966 etime=1364999966 start
    =1365018810 owner=mduser2@qgg.hud.ac.uk exec_host=enode18.eridani.qgg.hud.ac.uk/3+
    enode18.eridani.qgg.hud.ac.uk/2+enode18.eridani.qgg.hud.ac.uk/1+enode18.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=21176
     end=1365101660 Exit_status=0 resources_used.cput=21:57:14 resources_used.walltime
    =23:00:50 app_name=dlpoly iterations=5000000 atoms=500
```

```
04/04/2013 19:35:41;E;77775.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999921 qtime=1364999921 etime=1364999921 start
    =1365017691 owner=mduser2@qgg.hud.ac.uk exec_host=enode23.eridani.qgg.hud.ac.uk/3+
    enode23.eridani.qgg.hud.ac.uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=12655
     end=1365100541 Exit_status=0 resources_used.cput=24:57:36 resources_used.walltime
    =23:00:50 app_name=dlpoly iterations=5000000 atoms=500
04/04/2013 09:07:40;E;77738.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999320 qtime=1364999320 etime=1364999320 start
    =1364999320 owner=mduser2@qgg.hud.ac.uk exec_host=enode30.eridani.qgg.hud.ac.uk/3+
    enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=7671
    end=1365062860 Exit_status=0 resources_used.cput=70:31:43 resources_used.walltime
    =17:39:00 app_name=dlpoly iterations=5000000 atoms=500
04/04/2013 12:07:07;E;77768.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999749 qtime=1364999749 etime=1364999749 start
    =1365002242 owner=mduser2@qgg.hud.ac.uk exec_host=enode34.eridani.qgg.hud.ac.uk/3+
    enode34.eridani.qgg.hud.ac.uk/2+enode34.eridani.qgg.hud.ac.uk/1+enode34.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=8017
    end=1365073627 Exit_status=0 resources_used.cput=79:14:18 resources_used.walltime
    =19:49:45 app_name=dlpoly iterations=5000000 atoms=500
04/04/2013 12:28:22;E;77733.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=serialstd ctime=1364999241 qtime=1364999241 etime=1364999241 start
    =1364999241 owner=mduser2@qgg.hud.ac.uk exec_host=enode28.eridani.qgg.hud.ac.uk/3+
    enode28.eridani.qgg.hud.ac.uk/2+enode28.eridani.qgg.hud.ac.uk/1+enode28.eridani.
    qgg.hud.ac.uk/0 Resource_List.cput=192:00:00 Resource_List.ncpus=1 Resource_List.
    neednodes=1:ppn=4 Resource_List.nodect=1 Resource_List.nodes=1:ppn=4 session=8773
    end=1365074902 Exit_status=0 resources_used.cput=83:58:44 resources_used.walltime
    =21:01:01 app_name=dlpoly iterations=5000000 atoms=500
```

```
24/04/2013 12:06:21;E;101863.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=paraul ctime=1366640324 qtime=1366640324 etime=1366640324 start
    =1366640325 owner=mduser2@qgg.hud.ac.uk exec_host=enode35.eridani.qgg.hud.ac.uk/3+
    enode35.eridani.qgg.hud.ac.uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.eridani.
    qgg.hud.ac.uk/0+enode34.eridani.qgg.hud.ac.uk/3+enode34.eridani.qgg.hud.ac.uk/2+
    enode34.eridani.qgg.hud.ac.uk/1+enode34.eridani.qgg.hud.ac.uk/0+enode33.eridani.
    qgg.hud.ac.uk/3+enode33.eridani.qgg.hud.ac.uk/2+enode33.eridani.qgg.hud.ac.uk/1+
    enode33.eridani.qgg.hud.ac.uk/0+enode32.eridani.qgg.hud.ac.uk/3+enode32.eridani.
    qgg.hud.ac.uk/2+enode32.eridani.qgg.hud.ac.uk/1+enode32.eridani.qgg.hud.ac.uk/0+
    enode31.eridani.qgg.hud.ac.uk/3+enode31.eridani.qgg.hud.ac.uk/2+enode31.eridani.
    qgg.hud.ac.uk/1+enode31.eridani.qgg.hud.ac.uk/0+enode30.eridani.qgg.hud.ac.uk/3+
    enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.eridani.
    qgg.hud.ac.uk/0+enode29.eridani.qgg.hud.ac.uk/3+enode29.eridani.qgg.hud.ac.uk/2+
    enode29.eridani.qgg.hud.ac.uk/1+enode29.eridani.qgg.hud.ac.uk/0+enode28.eridani.
    qgg.hud.ac.uk/3+enode28.eridani.qgg.hud.ac.uk/2+enode28.eridani.qgg.hud.ac.uk/1+
    enode28.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List.ncpus
    =1 Resource_List.neednodes=8:ppn=4 Resource_List.nodect=8 Resource_List.nodes=8:
    ppn=4 session=29825 end=1366801581 Exit_status=0 resources_used.cput=11:22:01
    resources_used.walltime=44:47:36 app_name=dlpoly iterations=1000000 atoms=8000
23/04/2013 09:43:56;E;101864.eridani.qgg.hud.ac.uk;user=mduser2 group=mduser2 jobname=
    jcl queue=paraul ctime=1366706619 qtime=1366706619 etime=1366706619 start
    =1366706619 owner=mduser2@qgg.hud.ac.uk exec_host=enode35.eridani.qgg.hud.ac.uk/3+
    enode35.eridani.qgg.hud.ac.uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.eridani.
    qgg.hud.ac.uk/0+enode34.eridani.qgg.hud.ac.uk/3+enode34.eridani.qgg.hud.ac.uk/2+
    enode34.eridani.qgg.hud.ac.uk/1+enode34.eridani.qgg.hud.ac.uk/0+enode33.eridani.
    qgg.hud.ac.uk/3+enode33.eridani.qgg.hud.ac.uk/2+enode33.eridani.qgg.hud.ac.uk/1+
    enode33.eridani.qgg.hud.ac.uk/0+enode32.eridani.qgg.hud.ac.uk/3+enode32.eridani.
    qgg.hud.ac.uk/2+enode32.eridani.qgg.hud.ac.uk/1+enode32.eridani.qgg.hud.ac.uk/0+
    enode31.eridani.qgg.hud.ac.uk/3+enode31.eridani.qgg.hud.ac.uk/2+enode31.eridani.
    qgg.hud.ac.uk/1+enode31.eridani.qgg.hud.ac.uk/0+enode30.eridani.qgg.hud.ac.uk/3+
    enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.eridani.
    qgg.hud.ac.uk/0+enode29.eridani.qgg.hud.ac.uk/3+enode29.eridani.qgg.hud.ac.uk/2+
    enode29.eridani.qgg.hud.ac.uk/1+enode29.eridani.qgg.hud.ac.uk/0+enode28.eridani.
    qgg.hud.ac.uk/3+enode28.eridani.qgg.hud.ac.uk/2+enode28.eridani.qgg.hud.ac.uk/1+
    enode28.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List.ncpus
    =1 Resource_List.neednodes=8:ppn=4 Resource_List.nodect=8 Resource_List.nodes=8:
    ppn=4 session=30192 end=1366706636 Exit_status=0 resources_used.cput=00:00:00
    resources_used.walltime=00:00:17 app_name=dlpoly iterations=100 atoms=8000
```

```
02/04/2013 00:14:10;E;77182.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case1 queue=paraul ctime=1364858024 qtime=1364858024 etime=1364858024 start
    =1364858024 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode19.eridani.qgg.hud
    .ac.uk/3+enode19.eridani.qgg.hud.ac.uk/2+enode19.eridani.qgg.hud.ac.uk/1+enode19.
    eridani.qgg.hud.ac.uk/0+enode18.eridani.qgg.hud.ac.uk/3+enode18.eridani.qgg.hud.ac
    .uk/2+enode18.eridani.qgg.hud.ac.uk/1+enode18.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 session=18948 end=1364858050
     Exit_status=271 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
02/04/2013 00:13:57;E;77181.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case237 queue=paraul ctime=1364858011 qtime=1364858011 etime=1364858011 start
    =1364858011 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode17.eridani.qgg.hud
    .ac.uk/3+enode17.eridani.qgg.hud.ac.uk/2+enode17.eridani.qgg.hud.ac.uk/1+enode17.
    eridani.qgg.hud.ac.uk/0+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
    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 session=20312 end=1364858037
     Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
02/04/2013 00:16:20;E;77183.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case238 queue=paraul ctime=1364858154 qtime=1364858154 etime=1364858154 start
    =1364858154 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode17.eridani.qgg.hud
    .ac.uk/3+enode17.eridani.qgg.hud.ac.uk/2+enode17.eridani.qgg.hud.ac.uk/1+enode17.
    eridani.qgg.hud.ac.uk/0+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
    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 session=21285 end=1364858180
     Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
02/04/2013 00:16:27;E;77184.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case239 queue=paraul ctime=1364858161 qtime=1364858161 etime=1364858161 start
    =1364858161 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode19.eridani.qgg.hud
    .ac.uk/3+enode19.eridani.qgg.hud.ac.uk/2+enode19.eridani.qgg.hud.ac.uk/1+enode19.
    eridani.qgg.hud.ac.uk/0+enode18.eridani.qgg.hud.ac.uk/3+enode18.eridani.qgg.hud.ac
    .uk/2+enode18.eridani.qgg.hud.ac.uk/1+enode18.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 session=19641 end=1364858187
     Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
```

```
02/04/2013 00:16:33;E;77185.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case240 queue=paraul ctime=1364858167 qtime=1364858167 etime=1364858167 start
    =1364858167 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode22.eridani.qgg.hud
    .ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.
    eridani.qgg.hud.ac.uk/0+enode21.eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac
    .uk/2+enode21.eridani.qgg.hud.ac.uk/1+enode21.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 session=21144 end=1364858193
     Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
02/04/2013 00:16:38;E;77186.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case241 queue=paraul ctime=1364858172 qtime=1364858172 etime=1364858172 start
    =1364858172 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode24.eridani.qgg.hud
    .ac.uk/3+enode24.eridani.qgg.hud.ac.uk/2+enode24.eridani.qgg.hud.ac.uk/1+enode24.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.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 session=13325 end=1364858198
     Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
02/04/2013 00:16:48;E;77187.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case242 queue=paraul ctime=1364858182 qtime=1364858182 etime=1364858182 start
    =1364858182 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode26.eridani.qgg.hud
    .ac.uk/3+enode26.eridani.qgg.hud.ac.uk/2+enode26.eridani.qgg.hud.ac.uk/1+enode26.
    eridani.qgg.hud.ac.uk/0+enode25.eridani.qgg.hud.ac.uk/3+enode25.eridani.qgg.hud.ac
    .uk/2+enode25.eridani.qgg.hud.ac.uk/1+enode25.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 session=9402 end=1364858208
    Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
02/04/2013 13:57:59;E;77192.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case239 queue=paraul ctime=1364907453 qtime=1364907453 etime=1364907453 start
    =1364907453 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode22.eridani.qgg.hud
    .ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.
    eridani.qgg.hud.ac.uk/0+enode21.eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac
    .uk/2+enode21.eridani.qgg.hud.ac.uk/1+enode21.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 session=22282 end=1364907479
     Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
```

```
02/04/2013 13:58:15;E;77193.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case240 queue=paraul ctime=1364907469 qtime=1364907469 etime=1364907469 start
    =1364907469 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode24.eridani.qgg.hud
    .ac.uk/3+enode24.eridani.qgg.hud.ac.uk/2+enode24.eridani.qgg.hud.ac.uk/1+enode24.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.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 session=14463 end=1364907495
     Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
02/04/2013 13:58:27;E;77194.eridani.qgg.hud.ac.uk;user=cfduser1 group=pgr jobname=
    case241 queue=paraul ctime=1364907481 qtime=1364907481 etime=1364907481 start
    =1364907481 owner=cfduser1@eridani.qgg.hud.ac.uk exec_host=enode26.eridani.qgg.hud
    .ac.uk/3+enode26.eridani.qgg.hud.ac.uk/2+enode26.eridani.qgg.hud.ac.uk/1+enode26.
    eridani.qgg.hud.ac.uk/0+enode25.eridani.qgg.hud.ac.uk/3+enode25.eridani.qgg.hud.ac
    .uk/2+enode25.eridani.qgg.hud.ac.uk/1+enode25.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 session=10540 end=1364907507
     Exit_status=0 resources_used.cput=00:00:00 resources_used.walltime=00:00:26
    app_name=fluent iterations=1 elements=16000000
04/04/2013 01:53:43;E;78715.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365027644 qtime=1365027644 etime=1365027644 start
    =1365036667 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode35.eridani.qgg.hud
    .ac.uk/3+enode35.eridani.qgg.hud.ac.uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.
    eridani.qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac
    .uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.eridani.qgg.hud.ac.uk/0+enode21.
    eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac.uk/2+enode21.eridani.qgg.hud.ac
    .uk/1+enode21.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 session=23365 end=1365036823 Exit_status=0 resources_used.cput
    =00:00:01 resources_used.walltime=00:02:36 app_name=fluent iterations=12 elements
    =8100000
```

```
04/04/2013 11:29:59;E;79221.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365071243 qtime=1365071243 etime=1365071243 start
    =1365071243 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode23.eridani.qgg.hud
    .ac.uk/3+enode23.eridani.qgg.hud.ac.uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.
    eridani.qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac
    .uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.eridani.qgg.hud.ac.uk/0+enode21.
    eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac.uk/2+enode21.eridani.qgg.hud.ac
    .uk/1+enode21.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 session=14219 end=1365071399 Exit_status=0 resources_used.cput
    =00:00:01 resources_used.walltime=00:02:36 app_name=fluent iterations=12 elements
    =8100000
04/04/2013 11:43:40;E;79251.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365072064 qtime=1365072064 etime=1365072064 start
    =1365072064 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode30.eridani.qgg.hud
    .ac.uk/3+enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.eridani.qgg.hud.ac.uk/0+enode22.
    eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac
    .uk/1+enode22.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 session=8494 end=1365072220 Exit_status=0 resources_used.cput
    =00:00:06 resources_used.walltime=00:02:36 app_name=fluent iterations=12 elements
    =8100000
04/04/2013 12:20:48;E;79348.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365074292 qtime=1365074292 etime=1365074292 start
    =1365074292 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode30.eridani.qgg.hud
    .ac.uk/3+enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.eridani.qgg.hud.ac.uk/0+enode22.
    eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac
    .uk/1+enode22.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 session=9544 end=1365074448 Exit_status=0 resources_used.cput
    =00:00:00 resources_used.walltime=00:02:36 app_name=fluent iterations=12 elements
    =8100000
```

```
04/04/2013 12:27:22;E;79365.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365074685 qtime=1365074685 etime=1365074685 start
    =1365074686 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode30.eridani.qgg.hud
    .ac.uk/3+enode30.eridani.qgg.hud.ac.uk/2+enode30.eridani.qgg.hud.ac.uk/1+enode30.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.eridani.qgg.hud.ac.uk/0+enode22.
    eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac.uk/2+enode22.eridani.qgg.hud.ac
    .uk/1+enode22.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 session=10584 end=1365074842 Exit_status=0 resources_used.cput
    =00:00:00 resources_used.walltime=00:02:36 app_name=fluent iterations=12 elements
    =8100000
04/04/2013 12:40:08;E;79397.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365075452 qtime=1365075452 etime=1365075452 start
    =1365075452 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode23.eridani.qgg.hud
    .ac.uk/3+enode23.eridani.qgg.hud.ac.uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.
    eridani.qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac
    .uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.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 session=15483 end=1365075608
     Exit_status=0 resources_used.cput=00:00:01 resources_used.walltime=00:02:36
    app_name=fluent iterations=12 elements=8100000
04/04/2013 12:44:52;E;79409.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365075736 qtime=1365075736 etime=1365075736 start
    =1365075736 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode23.eridani.qgg.hud
    .ac.uk/3+enode23.eridani.qgg.hud.ac.uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.
    eridani.qgg.hud.ac.uk/0+enode22.eridani.qgg.hud.ac.uk/3+enode22.eridani.qgg.hud.ac
    .uk/2+enode22.eridani.qgg.hud.ac.uk/1+enode22.eridani.qgg.hud.ac.uk/0+enode21.
    eridani.qgg.hud.ac.uk/3+enode21.eridani.qgg.hud.ac.uk/2+enode21.eridani.qgg.hud.ac
    .uk/1+enode21.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 session=16452 end=1365075892 Exit_status=0 resources_used.cput
    =00:06:08 resources_used.walltime=00:02:36 app_name=fluent iterations=12 elements
    =8100000
```

```
04/04/2013 12:55:31;E;79432.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365076375 qtime=1365076375 etime=1365076375 start
    =1365076375 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode37.eridani.qgg.hud
    .ac.uk/3+enode37.eridani.qgg.hud.ac.uk/2+enode37.eridani.qgg.hud.ac.uk/1+enode37.
    eridani.qgg.hud.ac.uk/0+enode35.eridani.qgg.hud.ac.uk/3+enode35.eridani.qgg.hud.ac
    .uk/2+enode35.eridani.qgg.hud.ac.uk/1+enode35.eridani.qgg.hud.ac.uk/0+enode34.
    eridani.qgg.hud.ac.uk/3+enode34.eridani.qgg.hud.ac.uk/2+enode34.eridani.qgg.hud.ac
    .uk/1+enode34.eridani.qgg.hud.ac.uk/0 Resource_List.cput=10000:00:00 Resource_List
    .ncpus=1 Resource_List.neednodes=3:ppn=4 Resource_List.nodect=3 Resource_List.
    nodes=3:ppn=4 session=7069 end=1365076531 Exit_status=0 resources_used.cput
    =00:00:00 resources_used.walltime=00:02:36 app_name=fluent iterations=12 elements
    =8100000
04/04/2013 13:03:52;E;79462.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365076875 qtime=1365076876 etime=1365076876 start
    =1365076876 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode34.eridani.qgg.hud
    .ac.uk/3+enode34.eridani.qgg.hud.ac.uk/2+enode34.eridani.qgg.hud.ac.uk/1+enode34.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.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 session=8921 end=1365077032
    Exit_status=0 resources_used.cput=00:04:16 resources_used.walltime=00:02:36
    app_name=fluent iterations=12 elements=8100000
04/04/2013 13:34:19;E;79537.eridani.qgg.hud.ac.uk;user=cfduser2 group=pgr jobname
    =60000-0.262 queue=paraul ctime=1365078333 qtime=1365078333 etime=1365078333 start
    =1365078703 owner=cfduser2@eridani.qgg.hud.ac.uk exec_host=enode37.eridani.qgg.hud
    .ac.uk/3+enode37.eridani.qgg.hud.ac.uk/2+enode37.eridani.qgg.hud.ac.uk/1+enode37.
    eridani.qgg.hud.ac.uk/0+enode23.eridani.qgg.hud.ac.uk/3+enode23.eridani.qgg.hud.ac
    .uk/2+enode23.eridani.qgg.hud.ac.uk/1+enode23.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 session=8693 end=1365078859
    Exit_status=0 resources_used.cput=00:10:39 resources_used.walltime=00:02:36
    app_name=fluent iterations=12 elements=8100000
```

A.3 Normalised Data with Moulding Information

```
02/04/2013 00:37:43;E;77181.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case237 queue=
    paraul ctime=1364858011 qtime=1364859462 etime=1364858011 start=1364859462 end
    =1364859463 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
```

```
02/04/2013 00:37:44;E;77182.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case1 queue=
    paraul ctime=1364858024 qtime=1364859463 etime=1364858024 start=1364859463 end
    =1364859464 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:45;E;77183.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case238 queue=
    paraul ctime=1364858154 qtime=1364859464 etime=1364858154 start=1364859464 end
    =1364859465 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:46;E;77184.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case239 queue=
    paraul ctime=1364858161 qtime=1364859465 etime=1364858161 start=1364859465 end
    =1364859466 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:47;E;77185.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case240 queue=
    paraul ctime=1364858167 qtime=1364859466 etime=1364858167 start=1364859466 end
    =1364859467 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:48;E;77186.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case241 queue=
    paraul ctime=1364858172 qtime=1364859467 etime=1364858172 start=1364859467 end
    =1364859468 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:49;E;77187.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case242 queue=
    paraul ctime=1364858182 qtime=1364859468 etime=1364858182 start=1364859468 end
    =1364859469 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 13:57:34;E;77192.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case239 queue=
    paraul ctime=1364907453 qtime=1364907453 etime=1364907453 start=1364907453 end
    =1364907454 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 13:57:50;E;77193.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case240 queue=
    paraul ctime=1364907469 qtime=1364907469 etime=1364907469 start=1364907469 end
    =1364907470 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 13:58:02;E;77194.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case241 queue=
    paraul ctime=1364907481 qtime=1364907481 etime=1364907481 start=1364907481 end
    =1364907482 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
04/04/2013 14:28:11;E;77733.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999241 qtime=1364999241 etime=1364999241 start=1364999241 end
    =1365082091 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
```

```
04/04/2013 14:29:30;E;77738.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999320 qtime=1364999320 etime=1364999320 start=1364999320 end
    =1365082170 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:30:30;E;78715.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365027644 qtime=1365082132 etime=1365027644 start=1365082132
    end=1365082230 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:31:08;E;77747.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999418 qtime=1364999418 etime=1364999418 start=1364999418 end
    =1365082268 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:32:09;E;79221.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365071243 qtime=1365082231 etime=1365071243 start=1365082231
    end=1365082329 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:32:46;E;79251.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365072064 qtime=1365082268 etime=1365072064 start=1365082268
    end=1365082366 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:33:23;E;79348.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365074292 qtime=1365082305 etime=1365074292 start=1365082305
    end=1365082403 Resource_List.nodes=1:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:33:47;E;79365.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365074685 qtime=1365082329 etime=1365074685 start=1365082329
    end=1365082427 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:34:24;E;79397.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365075452 qtime=1365082366 etime=1365075452 start=1365082366
    end=1365082464 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:34:59;E;77761.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999649 qtime=1364999649 etime=1364999649 start=1364999649 end
    =1365082499 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:35:21;E;79409.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365075736 qtime=1365082423 etime=1365075736 start=1365082423
    end=1365082521 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
```

```
04/04/2013 14:36:39;E;77768.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999749 qtime=1364999749 etime=1364999749 start=1364999749 end
    =1365082599 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:37:12;E;79432.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365076375 qtime=1365082534 etime=1365076375 start=1365082534
    end=1365082632 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:39:31;E;77775.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999921 qtime=1364999921 etime=1364999921 start=1364999921 end
    =1365082771 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:40:16;E;77776.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999966 qtime=1364999966 etime=1364999966 start=1364999966 end
    =1365082816 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:41:02;E;77778.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1365000012 qtime=1365000012 etime=1365000012 start=1365000012 end
    =1365082862 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:41:09;E;79462.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365076875 qtime=1365082771 etime=1365076876 start=1365082771
    end=1365082869 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:42:47;E;79537.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365078333 qtime=1365082869 etime=1365078333 start=1365082869
    end=1365082967 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
24/04/2013 12:24:51;E;101864.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    paraul ctime=1366706619 qtime=1366802674 etime=1366706619 start=1366802674 end
    =1366802691 Resource_List.nodes=1:ppn=4 resources_used.walltime=00:00:17 app_name=
    dlpoly iterations=100 dataset_size=8000
26/04/2013 09:12:10;E;101863.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    paraul ctime=1366640324 qtime=1366802674 etime=1366640324 start=1366802674 end
    =1366963930 Resource_List.nodes=4:ppn=4 resources_used.walltime=44:47:36 app_name=
    dlpoly iterations=1000000 dataset_size=8000
27/04/2013 06:39:18;E;73002.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366889417 qtime=1366968592 etime=1366889417 start=1366968592 end
    =1367041158 Resource_List.nodes=4:ppn=4 resources_used.walltime=20:09:26 app_name=
    dlpoly iterations=900000 dataset_size=4000
```

```
27/04/2013 08:01:01;E;75246.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982454 qtime=1367046036 etime=1366982454 start=1367046036 end
    =1367046061 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:01:26;E;75247.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982456 qtime=1367046061 etime=1366982456 start=1367046061 end
    =1367046086 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:01:37;E;75250.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982457 qtime=1367046072 etime=1366982457 start=1367046072 end
    =1367046097 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:01:51;E;75249.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982457 qtime=1367046086 etime=1366982457 start=1367046086 end
    =1367046111 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:02:02;E;75248.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982457 qtime=1367046097 etime=1366982457 start=1367046097 end
    =1367046122 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:02:16;E;75251.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982458 qtime=1367046111 etime=1366982458 start=1367046111 end
    =1367046136 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:02:27;E;75252.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982459 qtime=1367046122 etime=1366982459 start=1367046122 end
    =1367046147 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 21:06:03;E;73207.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366897046 qtime=1367020597 etime=1366897046 start=1367020597 end
    =1367093163 Resource_List.nodes=4:ppn=4 resources_used.walltime=20:09:26 app_name=
    dlpoly iterations=900000 dataset_size=4000
27/04/2013 21:18:13;E;73269.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366898750 qtime=1367021327 etime=1366898750 start=1367021327 end
    =1367093893 Resource_List.nodes=4:ppn=4 resources_used.walltime=20:09:26 app_name=
    dlpoly iterations=900000 dataset_size=4000
```

Appendix B

Appendix B: Simulator

B.1 Code

```
#!/usr/bin/python
import sys, time
from string import ljust as tabf

# 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=1
if DEBUG3:
 DEBUG2=1
if DEBUG2:
 logger.setLevel(logging.DEBUG)
else:
 logger.setLevel(logging.ERROR)
logger.propagate = False

################################################################################
```

```
# Define the jobStruct class. This creates a structure to be used by the queues and
    the init_table.
# Populating it with Headings
class jobStruct:
 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 sortStruct class. This creates a structure to be used within the sorting
    queue.
# Just minimal information to track a jobs position in the system
class sortStruct:
 def __init__(self, trigger_time):
  self.trigger_times = trigger_time
  jobnos = ""
  run_OR_sort = ""

##############################################################################
## Function to read the resources file and populate the resource_table[] based on the
    number of cores avaliable.
## The resources.txt file consists of space seperated values node number (starting
    from 0) 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([])
   # now that the node is created see how many cores it has and build up the slots
   for i in range(0,int(temp[1])):
    resource_table[int(temp[0])].append(1)
  else:
   pass

 return resource_table #FIXME(needs error handling for misformated node definitions)

################################################################################
## Function to read the RAW torque logs (file names or paths to be sent by the
    function calling it). It added all the jobs as user
## generated input triggers for the system. This function also activates the queues as
     global arrays and counters to track lengths
def log_parse(torque_log):

 # The running_table depicts the half of the queue that has running jobs
 # Each array element will be of type jobStruct
 global running_table
 running_table = []

 # The queueing_table depicts the half of the queue that has queued up jobs
 # (due to lack of space on the system). Array element will be of type jobStruct
 global queueing_table
 queueing_table = []

 # The sorting table is an over all queue that is kept sorted to showq
 # the position of jobs in the system. Element type: sortStruct
 global sorting_table
 sorting_table = []

 # The init_table holds the jobs parsed from the logs and this table forms the
 # "user triggered" events. Array element will be of type jobStruct
 global init_table
 init_table = []
```

```
# Track length of input triggers
global init_tab_length
init_tab_length = 0

global previous_trigger

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(";")
 ## Only process E records from accounting log
 if temp0[1] == "E":
  # keep the fully formed job number
  fjobno = str(temp0[2])
  # split job number for numeric parts only and check for array jobs.
  # Array jobs handled as decimaled strings
  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]

  # split the remaining aspects of the line to get the job request properties
  temp1 = temp0[3].split(" ")

  for i in range(0, len(temp1)):
   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
  # check for "NULL" nodes or ppn values. If present set default values of 1,1
  if nodes == "NULL" or ppn == "NULL":
   nodes = 1
   ppn = 1

  duration = int(end) - int(start)

  # assign striped job nos, and job properties to a new "Job" and place into the
    init_table
  if find_space(int(nodes),int(ppn),jobno):
   init_table.append(jobStruct(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))

 # "Resource_List.nodes" is not always present. Set values to "NULL" to allow
    checking
 nodes = "NULL"
```

```
  ppn = "NULL"

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

 if DEBUG3:
  display_table("init_table")

################################################################################
## Originally based on -- 9-12-13 rosettacode.org/wiki/Sorting_algorithms/Bubble_sort#
    Python
## Function to sort array 'seq'. For init_table, sort on jobno. For sortng_table, sort
     on trigger_times.
def bubble_sort(seq,index):
 if index == 'init':
  changed = True
  while changed:
   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
 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

################################################################################
## Function to assign the resources required for a job to the resource_table
## If sucessful returns an array (made_busy) of resources assigned, otherwise retuns 0
def make_busy(nodes,ppn,jobnos):
 # Count of cpus allocated
 no_cpus = 0
 # Count of nodes allocated
 no_nodes = 0
 # Continuation flag to break first loop if only partial cpu requirements have been
    met.
 cont = True
```

```
# Array to hold values of which cores have been set to busy
global made_busy
made_busy = []

logger.debug("Making Busy for "+str(jobnos))
# Traverse resource_table finding nodes where all cores are free and number of cores
    exactly matches
# the number of cores (per node) required.
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
     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")
     # Exact match cannot be made, Do not continue within this loop
     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 all allocations have been made end function here
if no_nodes == int(nodes):
 print "Making Allocation: ",nodes,ppn," at: ",made_busy
 if DEBUG2:
  display_table("resource_table")
 return made_busy

# Where exact resource matches cannot be found. Traverse the resource table finding
    any nodes which
```

```
 # can accomodate the resources required for the job.
 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]
    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

 # Check if all allocations have been made. If not return 0, else return array of
    made_busy
 if no_nodes != int(nodes):
  return 0
 else:
  print "Making Allocation: ",nodes,ppn," at: ",made_busy
  if DEBUG2:
   display_table("resource_table")
  return made_busy

##############################################################################
## Function to release resources previously marked as used in the resource_table
## Takes an array arguement (busy_cores) of cordinates previousy returned from
    make_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

################################################################################
## Function to check if the resources required for a job are avaliable in the
    resource_table
## If sucessful returns 1, otherwise retuns 0
def find_space(nodes,ppn,jobnos):

 # Count of cpus allocated
 no_cpus = 0
 # Count of node allocated
 no_nodes = 0
 # Continuation flag to break first loop if only partial cpu requirements have been
    met.
 cont = True
 # Number of nodes which meet job requirements
 node_sum=0
 # Number of cores/per node which meet the job requirements
 core_sum=0


 logger.debug("Find Space for "+str(jobnos))
 # Traverse resource_table finding nodes where all cores are free and number of cores
    exactly matches
 # the number of cores (per node) required.
 for i in range(0,len(resource_table)):
  if len(resource_table[i]) >= ppn:
   node_sum=node_sum+1
 if node_sum>=nodes:

  # 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")
      # Exact match cannot be made, Do not continue within this loop and reset the
   cpu count
      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

# If sufficient resources have been found exit function here
if no_nodes == int(nodes):
 return 1

# Where exact resource matches cannot be found. Traverse the resource table finding
   any nodes which
# can accomodate the resources required for the job.
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]
   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

 # Check if all allocations have been made. If not return 0, else return 1
 if no_nodes != int(nodes):
  return 0
 else:
  return 1


################################################################################
## Function to convert values in seconds to Hours:Minutes:Seconds
def format_seconds_to_hhmmss(seconds):
 hours = seconds // (60*60)
 seconds %= (60*60)
 minutes = seconds // 60
 seconds %= 60
 return "%02i:%02i:%02i" % (hours, minutes, seconds)
################################################################################
## Function to print final job description to a torque style log file.
## This can be passed the result of a pop an any table using jobStruct.

def print_logs(completed_job):

 # Convert job epoch end time to a human redable string
 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("%s;E;%s;user=%s queue=%s ctime=%s qtime=%s etime=%s start=%s end=%s
     Resource_List.nodes=%s:ppn=%s resources_used.walltime=%s\n" % (tstamp,
     completed_job.fjobno, completed_job.username, completed_job.queue, completed_job.
     ctime, completed_job.qtime, completed_job.etime, completed_job.start,
     completed_job.end, completed_job.nodes, completed_job.ppn,
     format_seconds_to_hhmmss(completed_job.duration)))
 new_log.close()

################################################################################
## Function to move (COPY!!) job information between tables. Takes arguements source
     table, source table index, destnation table,
## resources (returned from make_busy) and a sort_flag. sort_flag is used to determine
      if the detination table is populated with
```

```
## the sortStruct and if the job should be marked as running "R" or queued "Q".
    Returns the index of the source table to which
## the job data was moved (COPIED!!)
def move_job(src_tab,src_index,dest_tab,resources,sort_flag):

 dest_rec_index = ""

 if sort_flag == 'R':
  dest_tab.append(sortStruct(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(sortStruct(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(jobStruct(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
  dest_tab[len(dest_tab)-1].fjobno = src_tab[src_index].fjobno
  # If placeholder value for resources is passed set to 0
  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

##############################################################################
## Function to find which indicies are populated by a particulr jobno. Takes
    arguements of job number and table to check.
## Returns index which matches job number
```

```
def find_index(jobno,table):

 for i in range(0,len(table)):
  if table[i].jobnos == jobno:
   return i

################################################################################
## Function which copies job information into the running_table. While also calling
    make busy to take resources.
## Takes arguements of index of source table (as returned by find_index), source table
     and value of previous_trigger (taken from
## previously popped job trigger_time). Uses previous_trigger to determine if start
    and end time of currently considered job needs
## to be modified. Returns index value for running_table for job which has been set to
     "running".
def make_running(i,table,previous_trigger):

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

 # If previous trigger had a value other than 0 eveluate correct start and end times
    based on this value.
 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 use creation and elegible time to calculate start and end
 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

 # Udpate values
 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

 if DEBUG2:
  display_table("running_table")
  display_table("queueing_table")
  display_table("sorting_table")

 return running_job_index

################################################################################
## Function which copies job information into the queueing_table. While also calling
     make busy to take resources.
## Takes argument of init_table index for job being considered.
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)
 if DEBUG2:
  display_table("running_table")
  display_table("queueing_table")
  display_table("sorting_table")
 return queued_job_index

################################################################################
## Function to process items in the queue. Takes an arguement of init_trigger, which
     can be any valid epoch time, to
## determine at which point jobs can be popped from the running_table or moved from
     queued to running.
def process_queue_items(init_trigger):
 global previous_trigger
 j=0
 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," from ",completed_job.resources
```

```
      print "
      ____________________________________________________________________________"
      sorting_table.pop(j)

      if DEBUG3:
       print "Calling inside while (poped jobs)"
       display_table("sorting_table")
      j=0
     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: ",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
      else:
       j += 1
    # All jobs removed from sorting table. End processing
    if len(sorting_table) == 0:
     break
   else:
    j += 1

 return 0

##############################################################################
# The display_table function takes as argument a string from "sorting_table|
    queueing_table|running_table|init_table|resources" and
# outputs a somewhat formated table to the screen. If there are no records in the
    tables (except resources) it just says so

def display_table(table):

 if table == "sorting_table":
  if len(sorting_table) == 0:
   print "Sorting table is empty"
  else:
```

```
    print "\n-------------------------------------sorting
     ---------------------------------------"
    print tabf("Trigger",12),tabf("Job No",12),tabf("Queue",12)
    for k in range(0,len(sorting_table)):
     print tabf(str(sorting_table[k].trigger_times),12),tabf(sorting_table[k].jobnos
     ,12),tabf(sorting_table[k].run_OR_sort,12)
    print "
     ------------------------------------------------------------------------------"
  elif table == "queueing_table":
   if len(queueing_table) == 0:
    print "Queued table is empty"
   else:
    print "\n-------------------------------------queueing
     ---------------------------------------"
    print tabf("Job No",12),tabf("ctime",12),tabf("nodes",12),tabf("ppn",12),tabf("
     Duration",12)
    for k in range(0,len(queueing_table)):
     print tabf(queueing_table[k].jobnos,12),tabf(str(queueing_table[k].ctime),12),tabf
     (str(queueing_table[k].nodes),12),tabf(str(queueing_table[k].ppn),12),tabf(str(
     queueing_table[k].duration),12)
    print "
     ------------------------------------------------------------------------------"
  elif table == "resource_table":
    print "\n-------------------------------------system
     ---------------------------------------"
    for k in range(0,len(resource_table)):
     print resource_table[k]
    print "
     ------------------------------------------------------------------------------"
  elif table == "init_table":
   if len(init_table) == 0:
    print "Initial table is empty"
   else:
    print "\n-------------------------------------inittab
     ---------------------------------------"
    print tabf("Job No",8),tabf("ctime",11),tabf("nodes",5),tabf("ppn",3),tabf("
     Duration",8),tabf("Start",11),tabf("End",11)
    for k in range(0,len(init_table)):
     print tabf(init_table[k].jobnos,8),tabf(str(init_table[k].ctime),11),tabf(str(
     init_table[k].nodes),5),tabf(str(init_table[k].ppn),3),tabf(str(init_table[k].
     duration),8),tabf(str(init_table[k].start),11),tabf(str(init_table[k].end),11)
    print "
     ------------------------------------------------------------------------------"
```

```
 else:
  if len(running_table) == 0:
   print "Running table is empty"
  else:
   print "\n----------running
    ----------"
   print tabf("Job No",8),tabf("ctime",11),tabf("nodes",5),tabf("ppn",3),tabf("
    Duration",8),tabf("Start",11),tabf("End",11),tabf("Alloc",11)
   for k in range(0,len(running_table)):
    print tabf(running_table[k].jobnos,8),tabf(str(running_table[k].ctime),11),tabf(
    str(running_table[k].nodes),5),tabf(str(running_table[k].ppn),3),tabf(str(
    running_table[k].duration),8),tabf(str(running_table[k].start),11),tabf(str(
    running_table[k].end),11),tabf(str(running_table[k].resources),11)
   print "
    ----------"




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

 resource_parse()
 log_parse(sys.argv[1])

 previous_trigger = 0

 bubble_sort(init_table,'init')

 for i in range(0,init_tab_length):
  print "\n
    ----------"
  print "New Job is: ",init_table[i].jobnos," at time ",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):
     done_sorting=process_queue_items(init_table[i].ctime)

     if len(sorting_table) == 0:
      break

    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 "Sorting table after the final user triggered qsub"
 display_table("sorting_table")
while len(sorting_table) > 0:
 process_queue_items(9999999999)

 if len(sorting_table) == 0:
  break

print "\n\n"

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

```
print "
    ____________________________________________________________________________"


display_table("running_table")

display_table("queueing_table")

display_table("sorting_table")

display_table("resource_table")
```

B.2 Verficiation Sheet

	Original							Calculated		Simulated			DURATION	Start	End
jobno	ctime	qtime	start	end	nodes	ppn	duration	start	end	start	end	duration	CHECK	Check	Check
88133	1365417743	1365417743	1365417743	1365822395	1	1	404652	1365417743	1365822395	1365417743	1365822395	404652	1	1	1
97083	1365670048	1365670048	1365670048	1365817705	1	1	147657	1365670048	1365817705	1365670048	1365817705	147657	1	1	1
97089	1365687427	1365687427	1365687427	1365828004	2	4	140577	1365687427	1365828004	1365687427	1365828004	140577	1	1	1
97091	1365687445	1365687445	1365787770	1365885472	2	4	97702	1365687445	1365785147	1365687445	1365785147	97702	1	1	1
97092	1365687464	1365687464	1365787771	1365889245	2	4	101474	1365687464	1365788938	1365687464	1365788938	101474	1	1	1
97235	1365758740	1365758740	1365782103	1365816772	1	1	34669	1365758740	1365793409	1365758740	1365793409	34669	1	1	1
99123	1365807296	1365807296	1365807469	1365807671	1	1	202	1365807296	1365807498	1365807296	1365807498	202	1	1	1
99124	1365807375	1365807375	1365807562	1365807757	1	1	195	1365807375	1365807570	1365807375	1365807570	195	1	1	1
99125	1365807402	1365807402	1365807583	1365807779	1	1	196	1365807402	1365807598	1365807402	1365807598	196	1	1	1
99126	1365807403	1365807403	1365807583	1365807780	1	1	197	1365807403	1365807600	1365807403	1365807600	197	1	1	1
99127	1365807407	1365807407	1365807583	1365807780	1	1	197	1365807407	1365807604	1365807407	1365807604	197	1	1	1
99128	1365807408	1365807408	1365807583	1365807780	1	1	197	1365807408	1365807605	1365807408	1365807605	197	1	1	1
99129	1365807408	1365807408	1365807583	1365807780	1	1	197	1365807408	1365807605	1365807408	1365807605	197	1	1	1
99130	1365807408	1365807408	1365807584	1365807784	1	1	200	1365807408	1365807608	1365807408	1365807608	200	1	1	1
99131	1365807408	1365807408	1365807584	1365807782	1	1	198	1365807408	1365807606	1365807408	1365807606	198	1	1	1
99132	1365807408	1365807409	1365807585	1365807782	1	1	197	1365807409	1365807606	1365807409	1365807606	197	1	1	1
99133	1365807412	1365807412	1365807585	1365807782	1	1	197	1365807412	1365807609	1365807412	1365807609	197	1	1	1
99134	1365807412	1365807413	1365807586	1365807783	1	1	197	1365807413	1365807610	1365807413	1365807610	197	1	1	1
99135	1365807412	1365807413	1365807586	1365807784	1	1	198	1365807413	1365807611	1365807413	1365807611	198	1	1	1
99136	1365807413	1365807413	1365807586	1365807783	1	1	197	1365807413	1365807610	1365807413	1365807610	197	1	1	1
99137	1365807413	1365807413	1365807586	1365807784	1	1	198	1365807413	1365807611	1365807413	1365807611	198	1	1	1
99138	1365807413	1365807413	1365807586	1365807784	1	1	198	1365807413	1365807611	1365807413	1365807611	198	1	1	1
99139	1365807413	1365807413	1365807586	1365807784	1	1	198	1365807413	1365807611	1365807413	1365807611	198	1	1	1
99140	1365807413	1365807413	1365807586	1365807784	1	1	198	1365807413	1365807611	1365807413	1365807611	198	1	1	1
99141	1365807413	1365807413	1365807586	1365807783	1	1	197	1365807413	1365807610	1365807413	1365807610	197	1	1	1
99142	1365807413	1365807413	1365807586	1365807784	1	1	198	1365807413	1365807611	1365807413	1365807611	198	1	1	1
99143	1365807492	1365807492	1365807672	1365807867	1	1	195	1365807492	1365807687	1365807492	1365807687	195	1	1	1
99144	1365807591	1365807591	1365807761	1365807956	1	1	195	1365807591	1365807786	1365807591	1365807786	195	1	1	1
99145	1365807599	1365807599	1365807783	1365807980	1	1	197	1365807599	1365807796	1365807599	1365807796	197	1	1	1
99146	1365807606	1365807606	1365807783	1365807980	1	1	197	1365807606	1365807803	1365807606	1365807803	197	1	1	1
99147	1365807606	1365807606	1365807783	1365807981	1	1	198	1365807606	1365807804	1365807606	1365807804	198	1	1	1
99148	1365807606	1365807606	1365807783	1365807989	1	1	206	1365807606	1365807812	1365807606	1365807812	206	1	1	1
99149	1365807607	1365807607	1365807783	1365807981	1	1	198	1365807607	1365807805	1365807607	1365807805	198	1	1	1
99150	1365807647	1365807647	1365807783	1365807980	1	1	197	1365807647	1365807844	1365807647	1365807844	197	1	1	1
99151	1365807647	1365807647	1365807783	1365807979	1	1	196	1365807647	1365807843	1365807647	1365807843	196	1	1	1
99152	1365807647	1365807647	1365807783	1365807980	1	1	197	1365807647	1365807844	1365807647	1365807844	197	1	1	1
99153	1365807647	1365807647	1365807784	1365807988	1	1	204	1365807647	1365807851	1365807647	1365807851	204	1	1	1
99154	1365807647	1365807647	1365807783	1365807981	1	1	198	1365807647	1365807845	1365807647	1365807845	198	1	1	1
99155	1365807647	1365807647	1365807784	1365807989	1	1	205	1365807647	1365807852	1365807647	1365807852	205	1	1	1
99156	1365807650	1365807650	1365807784	1365807989	1	1	205	1365807650	1365807855	1365807650	1365807855	205	1	1	1
99157	1365807650	1365807650	1365807784	1365807989	1	1	205	1365807650	1365807855	1365807650	1365807855	205	1	1	1
99158	1365807656	1365807656	1365807785	1365807988	1	1	203	1365807656	1365807859	1365807656	1365807859	203	1	1	1
99159	1365807656	1365807656	1365807785	1365807989	1	1	204	1365807656	1365807860	1365807656	1365807860	204	1	1	1
99160	1365807656	1365807656	1365807785	1365807989	1	1	204	1365807656	1365807860	1365807656	1365807860	204	1	1	1
99161	1365807665	1365807665	1365807785	1365807989	1	1	204	1365807665	1365807869	1365807665	1365807869	204	1	1	1
99162	1365807665	1365807665	1365807785	1365807988	1	1	203	1365807665	1365807868	1365807665	1365807868	203	1	1	1
99163	1365807695	1365807695	1365807876	1365808071	1	1	195	1365807695	1365807890	1365807695	1365807890	195	1	1	1
99164	1365807797	1365807797	1365807961	1365808156	1	1	195	1365807797	1365807992	1365807797	1365807992	195	1	1	1
99165	1365807804	1365807804	1365807982	1365808178	1	1	196	1365807804	1365808000	1365807804	1365808000	196	1	1	1
99166	1365807805	1365807805	1365807982	1365808181	1	1	199	1365807805	1365808004	1365807805	1365808004	199	1	1	1
99167	1365807805	1365807805	1365807982	1365808179	1	1	197	1365807805	1365808002	1365807805	1365808002	197	1	1	1
99168	1365807805	1365807805	1365807982	1365808179	1	1	197	1365807805	1365808002	1365807805	1365808002	197	1	1	1
99169	1365807805	1365807805	1365807982	1365808179	1	1	197	1365807805	1365808002	1365807805	1365808002	197	1	1	1
99170	1365807805	1365807805	1365807983	1365808179	1	1	196	1365807805	1365808001	1365807805	1365808001	196	1	1	1
99171	1365807809	1365807809	1365807983	1365808179	1	1	196	1365807812	1365808008	1365807812	1365808008	196	1	1	1
99172	1365807809	1365807809	1365807983	1365808179	1	1	196	1365807843	1365808039	1365807843	1365808039	196	1	1	1
99173	1365807809	1365807809	1365807988	1365808185	1	1	197	1365807844	1365808041	1365807844	1365808041	197	1	1	1
99174	1365807810	1365807810	1365807989	1365808185	1	1	196	1365807844	1365808040	1365807844	1365808040	196	1	1	1
99175	1365807810	1365807810	1365807989	1365808186	1	1	197	1365807845	1365808042	1365807845	1365808042	197	1	1	1
99176	1365807813	1365807813	1365807990	1365808186	1	1	196	1365807851	1365808047	1365807851	1365808047	196	1	1	1
99177	1365807813	1365807813	1365807989	1365808186	1	1	197	1365807852	1365808049	1365807852	1365808049	197	1	1	1
99178	1365807813	1365807813	1365807990	1365808186	1	1	196	1365807855	1365808051	1365807855	1365808051	196	1	1	1
99179	1365807813	1365807813	1365807990	1365808186	1	1	196	1365807855	1365808051	1365807855	1365808051	196	1	1	1
99180	1365807818	1365807818	1365807990	1365808186	1	1	196	1365807859	1365808055	1365807859	1365808055	196	1	1	1
99181	1365807818	1365807818	1365807990	1365808186	1	1	196	1365807860	1365808056	1365807860	1365808056	196	1	1	1
99182	1365807820	1365807820	1365807990	1365808185	1	1	195	1365807860	1365808055	1365807860	1365808055	195	1	1	1
99183	1365807899	1365807899	1365808074	1365808275	1	1	201	1365807899	1365808100	1365807899	1365808100	201	1	1	1
99184	1365807975	1365807975	1365808162	1365808359	1	1	197	1365807975	1365808172	1365807975	1365808172	197	1	1	1
99185	1365808006	1365808006	1365808182	1365808384	1	1	202	1365808006	1365808208	1365808006	1365808208	202	1	1	1
99186	1365808014	1365808014	1365808182	1365808384	1	1	202	1365808014	1365808216	1365808014	1365808216	202	1	1	1
99187	1365808014	1365808014	1365808183	1365808384	1	1	201	1365808014	1365808215	1365808014	1365808215	201	1	1	1
99188	1365808014	1365808014	1365808183	1365808384	1	1	201	1365808014	1365808215	1365808014	1365808215	201	1	1	1
99189	1365808014	1365808014	1365808183	1365808383	1	1	200	1365808014	1365808214	1365808014	1365808214	200	1	1	1
99190	1365808014	1365808014	1365808183	1365808383	1	1	200	1365808014	1365808214	1365808014	1365808214	200	1	1	1
99191	1365808014	1365808014	1365808183	1365808384	1	1	201	1365808014	1365808215	1365808014	1365808215	201	1	1	1
99192	1365808014	1365808014	1365808183	1365808384	1	1	201	1365808014	1365808215	1365808014	1365808215	201	1	1	1
99193	1365808018	1365808018	1365808186	1365808386	1	1	200	1365808018	1365808218	1365808018	1365808218	200	1	1	1
99194	1365808018	1365808018	1365808186	1365808383	1	1	197	1365808039	1365808236	1365808039	1365808236	197	1	1	1
99195	1365808018	1365808018	1365808186	1365808383	1	1	197	1365808040	1365808237	1365808040	1365808237	197	1	1	1
99196	1365808018	1365808018	1365808188	1365808385	1	1	197	1365808041	1365808238	1365808041	1365808238	197	1	1	1
99197	1365808018	1365808018	1365808188	1365808385	1	1	197	1365808042	1365808239	1365808042	1365808239	197	1	1	1
99198	1365808018	1365808018	1365808188	1365808385	1	1	197	1365808047	1365808244	1365808047	1365808244	197	1	1	1
99199	1365808018	1365808018	1365808188	1365808385	1	1	197	1365808049	1365808246	1365808049	1365808246	197	1	1	1
99200	1365808018	1365808018	1365808188	1365808385	1	1	197	1365808051	1365808248	1365808051	1365808248	197	1	1	1
99201	1365808018	1365808018	1365808188	1365808385	1	1	197	1365808051	1365808248	1365808051	1365808248	197	1	1	1
99202	1365808018	1365808018	1365808188	1365808384	1	1	196	1365808055	1365808251	1365808055	1365808251	196	1	1	1
99203	1365808097	1365808097	1365808283	1365808478	1	1	195	1365808097	1365808292	1365808097	1365808292	195	1	1	1
99204	1365808186	1365808186	1365808366	1365808569	1	1	203	1365808186	1365808389	1365808186	1365808389	203	1	1	1
99205	1365808196	1365808196	1365808387	1365808593	1	1	206	1365808196	1365808402	1365808196	1365808402	206	1	1	1
99206	1365808202	1365808202	1365808387	1365808593	1	1	206	1365808202	1365808408	1365808202	1365808408	206	1	1	1
99207	1365808202	1365808202	1365808387	1365808593	1	1	206	1365808208	1365808414	1365808208	1365808414	206	1	1	1

FIGURE B.1: Manual Verification of Simulated logs - truncated

Appendix C

Appendix C: Mouldable Scheduler

C.1 Code

```
#!/usr/bin/python
import sys, time#, sched_common as schcom
import MySQLdb as mdb
from string import ljust as tabf
from sched_common import display_table,sortStruct

import logging
logger = logging.getLogger('mould_job')
hdlr = logging.FileHandler('mould_job.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)

DEBUG2=0



class mouldStruct:
 def __init__(self, jobno):
  self.jobnos = jobno
  fjobno =""
  user = ""
  jobname = ""
```

```
  ctime = 0
  app_name = ""
  workload_size = 0
  dataset_size = 0
  start = 0
  end = 0
  nodes = 0
  ppn = 0
  duration = 0
  requested_nodes = 0
  requested_ppn = 0
  orig_duration = 0
  queue = ""
  qtime = 0
  etime = 0
  resources = []
  bmData = []

################################################################################
# Define the jobStruct class. This creates a structure to be used by the queues and
    the init_table.
# Populating it with Headings
class jobStruct:
 def __init__(self, jobno):
  self.jobnos = jobno
  fjobno = ""
  user = ""
  jobname = ""
  ctime = 0
  app_name = ""
  workload_size = 0
  dataset_size = 0
  start = 0
  end = 0
  nodes = 0
  ppn = 0
  duration = 0
  requested_nodes = 0
  requested_ppn = 0
  orig_duration = 0
  queue = ""
  qtime = 0
  etime = 0
```

```
  resources = []
  bmData = []

def get_job_info(torque_log):

 # The running_table depicts the half of the queue that has running jobs
 # Each array element will be of type jobStruct
 global running_table
 running_table = []

 # The queueing_table depicts the half of the queue that has queued up jobs
 # (due to lack of space on the system). Array element will be of type jobStruct
 global queueing_table
 queueing_table = []

 # The sorting table is an over all queue that is kept sorted to showq
 # the position of jobs in the system. Element type: sortStruct
 global sorting_table
 sorting_table = []

 global mould_table
 mould_table = []
 global mould_tab_length
 mould_tab_length = 0
 global con

 #app_name = "fluent"

 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("get_job_info: Populating the mould_table")

 con = mdb.connect('192.168.1.78','schede','HUD2010eng','aspp')
 with con:
  cur = con.cursor()
  for line in log_file:
   requested_nodes = 0
```

```
requested_ppn = 0
start = 0
end = 0
app_name = "NULL"
temp0 = line.split(";")
## Only process E records from accounting log
if temp0[1] == "E":
 # keep the fully formed job number
 fjobno = str(temp0[2])
 # split job number for numeric parts only and check for array jobs.
 # Array jobs handled as decimaled strings
 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]

 # split the remaining aspects of the line to get the job request properties
 temp1 = temp0[3].split(" ")

 for i in range(0, len(temp1)):
  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("jobname"):
   jobname = 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("app_name"):
   app_name = str(temp1[i].split("=")[1])
 ## Get orignal request data
  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(":")
    requested_nodes = int(temp2[0].split("=")[1])
    if len(temp2) == 2:
     requested_ppn = int(temp2[1].split("=")[1])
    else:
     requested_ppn = int(1)
   else:
    temp = temp1[i].split("=")
    sqlcmd = "SELECT * FROM applications WHERE appName='%s' and workMeta='%s'" % (
  app_name, temp[0])
    if cur.execute(sqlcmd):
     workload_size=temp[1].rstrip('\n')
    sqlcmd = "SELECT * FROM applications WHERE appName='%s' and dataMeta='%s'" % (
  app_name, temp[0])
    if cur.execute(sqlcmd):
     dataset_size=temp[1].rstrip('\n')

  ## check for "NULL" orignal nodes or ppn values. If present set default values of
  1,1
  if start == 0 or end == 0:
   orig_duration=0
  else:
   orig_duration = int(end) - int(start)

 mould_table.append(mouldStruct(jobno))
 mould_table[mould_tab_length].fjobno = fjobno
 mould_table[mould_tab_length].user = username
 mould_table[mould_tab_length].jobname = jobname
 mould_table[mould_tab_length].queue = queue
 mould_table[mould_tab_length].ctime = int(ctime)
 mould_table[mould_tab_length].qtime = int(qtime)
 mould_table[mould_tab_length].etime = int(etime)
 mould_table[mould_tab_length].app_name = app_name
 mould_table[mould_tab_length].start = 0
 mould_table[mould_tab_length].end = 0
 ##
 if app_name == "NULL":
  mould_table[mould_tab_length].requested_nodes = int(requested_nodes)
  mould_table[mould_tab_length].requested_ppn = int(requested_ppn)
  mould_table[mould_tab_length].orig_duration = int(orig_duration)
 else:
  requested_nodes=int(0)
  requested_ppn=int(0)
```

```
    orig_duration=int(0)
    mould_table[mould_tab_length].requested_nodes = int(requested_nodes)
    mould_table[mould_tab_length].requested_ppn = int(requested_ppn)
    mould_table[mould_tab_length].orig_duration = int(orig_duration)


   if int(requested_nodes) < 1:
    try:
     mould_table[mould_tab_length].workload_size = workload_size
    except UnboundLocalError:
     print "Bad workload Value"
     sys.exit(1)
    try:
     mould_table[mould_tab_length].dataset_size = dataset_size
    except UnboundLocalError:
     print "Bad dataset Value"
     sys.exit(1)
    # get node config and duration from the MySQL DB
    nodes, ppn, duration, bmData = initialMould(app_name,workload_size,dataset_size)
   else:
    nodes=requested_nodes
    ppn=requested_ppn
    duration=orig_duration
    mould_table[mould_tab_length].workload_size=0
    mould_table[mould_tab_length].dataset_size=0
    bmData=[]

   mould_table[mould_tab_length].nodes = nodes
   mould_table[mould_tab_length].ppn = ppn
   mould_table[mould_tab_length].duration = duration
   mould_table[mould_tab_length].bmData = bmData

   mould_tab_length = mould_tab_length + 1
   if int(requested_nodes) < 1:
    del workload_size,dataset_size

logger.debug("get_job_info: Completed the mould_table")


if len(mould_table) == 0:
 print "Moulding table is empty"
#else:
```

```
 # print "\n----------------------------------moulding
    ----------------------------------"
 # print tabf("Job No",8),tabf("User",8),tabf("ctime",11),tabf("App",10),tabf("
    Workload",10),tabf("Dataset",10),tabf("Nodes",5),tabf("Ppn",5),tabf("Duration",10)
    ,tabf("Benchmark Data",15)
 # for k in range(0,len(mould_table)):
 #  print tabf(mould_table[k].jobnos,8),tabf(mould_table[k].user,8),tabf(str(
    mould_table[k].ctime),11),tabf(mould_table[k].app_name,10),tabf(str(mould_table[k
    ].workload_size),10),tabf(str(mould_table[k].dataset_size),10),tabf(str(
    mould_table[k].nodes),5),tabf(str(mould_table[k].ppn),5),tabf(str(mould_table[k].
    duration),10),tabf(str(mould_table[k].bmData),15)

def bubble_sort(seq,index):
 if index == 'mould':
  changed = True
  while changed:
   changed = False
   for i in range(len(seq) - 1):
    if float(seq[i].ctime) > float(seq[i+1].ctime):
    #if float(seq[i].jobnos) > float(seq[i+1].jobnos):
     seq[i], seq[i+1] = seq[i+1], seq[i]
     changed = True
 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
 elif index == 'end':
  changed = True
  while changed:
   changed = False
   for i in range(len(seq) - 1):
    if seq[i].end > seq[i+1].end:
     seq[i], seq[i+1] = seq[i+1], seq[i]
     changed = True
 else:
  changed = True
  while changed:
   changed = False
   for i in range(len(seq) - 1):
```

```
    if seq[i].duration > seq[i+1].duration:
     seq[i], seq[i+1] = seq[i+1], seq[i]
     changed = True
 return seq

class bmStruct:
 def __init__(self, core):
  self.cores = core
  duration = 0

def initialMould(app_name,workload_size,dataset_size):

 bm_table = []
 cur=con.cursor()

 sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s' and
    dataset='%s'" % (app_name, workload_size,dataset_size)
 cur.execute(sqlcmd)
 rows = cur.fetchall()

 if rows:
  #the requested workload and dataset matched.
  #make a table of benchmark info that can be sorted to identify the optimum
     allocation for the perfect dS and wL match
  for i in range(0,len(rows)):
   bm_table.append(bmStruct(rows[i][0]))
   bm_table[i].duration = rows[i][1]

 else:
  #identify closest datasets for the app and workload specified.
  sqlcmd = "SELECT MIN(CASE WHEN dataset > '%s' AND appName='%s' AND workload='%s'
    THEN dataset ELSE NULL END) AS hgValue, MAX(CASE WHEN dataset < '%s' AND appName
    ='%s' AND workload='%s' THEN dataset ELSE NULL END) AS lowValue FROM benchmarks"
    % (dataset_size, app_name, workload_size, dataset_size, app_name, workload_size)
  cur.execute(sqlcmd)
  rows = cur.fetchall()

  if (rows[0][0] != None) or (rows[0][1] != None):

   # workload matches but dataset doesnot (#FIXME)
   dr =[]
```

```
# D1 is the larger dataset, D0 is the smaller dataset in comparison to the
 requested dataset DR
D1 = rows[0][0]
D0 = rows[0][1]

if (rows[0][0] != None) and (rows[0][1] != None):
# find the distance factor between the dataset sizes
 dist_factor=((float(dataset_size) - float(D0)) / (float(D1) - float(D0)))
else:
 if (D1 == None):
  dist_factor=float(dataset_size)/float(D0)
 else:
  dist_factor=float(dataset_size)/float(D1)
# get the cores and corresponding runtimes for the datasets D0 and D1
if (D0 != None):
 sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s'
 and dataset='%s'" % (app_name, workload_size,D0)
 cur.execute(sqlcmd)
 rowsD0 = cur.fetchall()
else:
 rowsD0 = ()

if (D1 != None):
 sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s'
 and dataset='%s'" % (app_name, workload_size,D1)
 cur.execute(sqlcmd)
 rowsD1 = cur.fetchall()
else:
 rowsD1 = ()

# for DR assign all cores and corresponding estimated runtimes.
for i in rowsD0:
 for j in rowsD1:
  if i[0] == j[0]:
   dtemp = (i[0]),int(round(0.4999+i[1]+(dist_factor*(j[1]-i[1]))))
   dr.append(dtemp)

# find those core values in the smaller dataset that have no corresponding values
 in the larger dataset
for i in rowsD0:
 found=0
 for j in rowsD1:
  if i[0] == j[0]:
```

```
      found=1
    if found==0:
     if (D1 != None):
      dtemp = (i[0]),int(round(0.4999+i[1]+(dist_factor*(i[1]))))
     else:
      dtemp = (i[0]),int(round(0.4999+(dist_factor*(i[1]))))
     dr.append(dtemp)

  # find those core values in the larger dataset that have no corresponding values in
     the smaller dataset
  for j in rowsD1:
   found=0
   for i in rowsD0:
    if i[0] == j[0]:
     found=1
   if found==0:
    if (D0 == None):
     dtemp = (j[0]),int(round(0.4999+(dist_factor*(j[1]))))
    else:
     dtemp = (j[0]),int(round(0.4999+((1-dist_factor)*(j[1]))))

    dr.append(dtemp)

  # with the bm data for non matching DS calculated, update the table of benchmarking
     data
  rows=dr

  #make a table of benchmark info that can be sorted to identify the optimum
    allocation for the non matching DS values
  for i in range(0,len(dr)):
   bm_table.append(bmStruct(dr[i][0]))
   bm_table[i].duration = dr[i][1]

 else:
  #identify closest datasets for the app and workload specified.
  sqlcmd = "SELECT MIN(CASE WHEN workload > '%s' AND appName='%s' AND dataset='%s'
   THEN workload ELSE NULL END) AS hgValue, MAX(CASE WHEN workload < '%s' AND appName
   ='%s' AND dataset='%s' THEN workload ELSE NULL END) AS lowValue FROM benchmarks"
   % (workload_size, app_name, dataset_size, workload_size, app_name, dataset_size)
  cur.execute(sqlcmd)
  rows = cur.fetchall()

  if (rows[0][0] != None) or (rows[0][1] != None):
```

```
# workload matches but dataset doesnot (#FIXME)
wr =[]

# W1 is the larger dataset, W0 is the smaller dataset in comparison to the
requested dataset wr
W1 = rows[0][0]
W0 = rows[0][1]

# find the distance factor between the dataset sizes
if (rows[0][0] != None) and (rows[0][1] != None):
 dist_factor=((float(workload_size) - float(W0)) / (float(W1) - float(W0)))
else:
 if (W1 == None):
  dist_factor=float(workload_size)/float(W0)
 else:
  dist_factor=float(workload_size)/float(W1)

# get the cores and corresponding runtimes for the datasets W0 and W1
if (W0 != None):
 sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s'
and dataset='%s'" % (app_name, W0, dataset_size)
 cur.execute(sqlcmd)
 rowsW0 = cur.fetchall()
else:
 rowsW0 = ()

if (W1 != None):
 sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s'
and dataset='%s'" % (app_name, W1, dataset_size)
 cur.execute(sqlcmd)
 rowsW1 = cur.fetchall()
else:
 rowsW1 = ()

# for WR assign all cores and corresponding estimated runtimes.
for i in rowsW0:
 for j in rowsW1:
  if i[0] == j[0]:
   dtemp = (i[0]),int(round(0.4999+i[1]+(dist_factor*(j[1]-i[1]))))
   wr.append(dtemp)
```

```
    # find those core values in the smaller dataset that have no corresponding values
    in the larger dataset
    for i in rowsW0:
     found=0
     for j in rowsW1:
      if i[0] == j[0]:
       found=1
     if found==0:
      dtemp = (i[0]),int(round(0.4999+(dist_factor*(i[1])))) #i[1]+
      wr.append(dtemp)

    # find those core values in the larger dataset that have no corresponding values
    in the smaller dataset
    for j in rowsW1:
     found=0
     for i in rowsW0:
      if i[0] == j[0]:
       found=1
     if found==0:
      dtemp = (j[0]),int(round(0.4999+(dist_factor*(j[1]))))
      wr.append(dtemp)

    # with the bm data for non matching DS calculated, update the table of
    benchmarking data
    rows=wr

    #make a table of benchmark info that can be sorted to identify the optimum
    allocation for the non matching DS values
    for i in range(0,len(wr)):
     bm_table.append(bmStruct(wr[i][0]))
     bm_table[i].duration = wr[i][1]
   else:
    drw0=[]
    drw1=[]
    drwr=[]
    my2dnone = [None,None]
    sqlcmd = "SELECT MIN(CASE WHEN workload > '%s' AND appName='%s' AND dataset > '%s'
     THEN workload ELSE NULL END) AS hgValue, MAX(CASE WHEN workload < '%s' AND
    appName='%s' AND dataset < '%s' THEN workload ELSE NULL END) AS lowValue FROM
    benchmarks" % (workload_size, app_name, dataset_size, workload_size, app_name,
    dataset_size)
    cur.execute(sqlcmd)
    rows_W4D = cur.fetchall()
```

```
if (rows_W4D[0][0] == None) and (rows_W4D[0][1] == None):

 sqlcmd = "SELECT MIN(CASE WHEN workload > '%s' AND appName='%s' AND dataset < '%s
' THEN workload ELSE NULL END) AS hgValue, MAX(CASE WHEN workload < '%s' AND
appName='%s' AND dataset > '%s' THEN workload ELSE NULL END) AS lowValue FROM
benchmarks" % (workload_size, app_name, dataset_size, workload_size, app_name,
dataset_size)
 cur.execute(sqlcmd)
 rows_W4D = cur.fetchall()
 #print rows_W4D

if (rows_W4D[0][0] != None):
 sqlcmd = "SELECT MIN(CASE WHEN dataset > '%s' AND appName='%s' AND workload='%s'
 THEN dataset ELSE NULL END) AS hgValue, MAX(CASE WHEN dataset < '%s' AND appName
='%s' AND workload='%s' THEN dataset ELSE NULL END) AS lowValue FROM benchmarks"
% (dataset_size, app_name, rows_W4D[0][0], dataset_size, app_name, rows_W4D[0][0])
 cur.execute(sqlcmd)
 rows_D4W1 = cur.fetchall()
else:
 rows_D4W1 = []
 rows_D4W1.append(my2dnone)

if (rows_W4D[0][1] != None):
 sqlcmd = "SELECT MIN(CASE WHEN dataset > '%s' AND appName='%s' AND workload='%s'
 THEN dataset ELSE NULL END) AS hgValue, MAX(CASE WHEN dataset < '%s' AND appName
='%s' AND workload='%s' THEN dataset ELSE NULL END) AS lowValue FROM benchmarks"
% (dataset_size, app_name, rows_W4D[0][1], dataset_size, app_name, rows_W4D[0][1])
 cur.execute(sqlcmd)
 rows_D4W0 = cur.fetchall()
else:
 rows_D4W0 = []
 rows_D4W0.append(my2dnone)

if (rows_W4D[0][0] != None):
 W1 = rows_W4D[0][0]
else:
 W1 = 0

if (rows_W4D[0][1] != None):
 W0 = rows_W4D[0][1]
else:
 W0 = 0
```

```
if (rows_D4W1[0][0] != None):
 D1W1 = rows_D4W1[0][0]
else:
 D1W1 = 0

if (rows_D4W1[0][1] != None):
 D0W1 = rows_D4W1[0][1]
else:
 D0W1 = 0

if (rows_D4W0[0][0] != None):
 D1W0 = rows_D4W0[0][0]
else:
 D1W0 = 0

if (rows_D4W0[0][1] != None):
 D0W0 = rows_D4W0[0][1]
else:
 D0W0 = 0

# find the distance factor between the dataset sizes
#print D0W0,D1W0,D0W1,D1W1
if (W1 != 0):
 dS_df_W1=abs((float(dataset_size) - float(D0W1)) / (float(D1W1) - float(D0W1)))
if (W0 != 0):
 dS_df_W0=abs((float(dataset_size) - float(D0W0)) / (float(D1W0) - float(D0W0)))
dist_factor=abs((float(workload_size) - float(W0)) / (float(W1) - float(W0)))

#
# get the 2 lines that correspond to W1 and reduce in a single line drw1
#

# get the cores and corresponding runtimes for the datasets D0W1 and D1W1
sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s'
and dataset='%s'" % (app_name, W1,D0W1)
cur.execute(sqlcmd)
rowsD0W1 = cur.fetchall()
sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s'
and dataset='%s'" % (app_name, W1,D1W1)
cur.execute(sqlcmd)
rowsD1W1 = cur.fetchall()
```

```
# for DR assign all cores and corresponding estimated runtimes IN W1.
for i in rowsD0W1:
 for j in rowsD1W1:
  if i[0] == j[0]:
   dtemp = (i[0]),i[1]+(dS_df_W1*(j[1]-i[1]))
   drw1.append(dtemp)

# find those core values in the smaller dataset that have no corresponding values
in the larger dataset
for i in rowsD0W1:
 found=0
 for j in rowsD1W1:
  if i[0] == j[0]:
   found=1
 if found==0:
  dtemp = (i[0]),i[1]+(dS_df_W1*(i[1]))
  drw1.append(dtemp)

# find those core values in the larger dataset that have no corresponding values
in the smaller dataset
for j in rowsD1W1:
 found=0
 for i in rowsD0W1:
  if i[0] == j[0]:
   found=1
 if found==0:
  if (D0W1 == 0):
   dtemp = (j[0]),(dS_df_W1*(j[1]))
  else:
   dtemp = (j[0]),((1-dS_df_W1)*(j[1]))

  drw1.append(dtemp)

#
# get the 2 lines that correspond to W0 and reduce in a single line drw0
#

# get the cores and corresponding runtimes for the datasets D0W0 and D1W0
sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s'
and dataset='%s'" % (app_name, W0,D0W0)
cur.execute(sqlcmd)
rowsD0W0 = cur.fetchall()
```

```
sqlcmd = "SELECT cores,time FROM benchmarks WHERE appName='%s' and workload='%s'
and dataset='%s'" % (app_name, W0,D1W0)
cur.execute(sqlcmd)
rowsD1W0 = cur.fetchall()

# for DR assign all cores and corresponding estimated runtimes IN W0.
for i in rowsD0W0:
 for j in rowsD1W0:
  if i[0] == j[0]:
   dtemp = (i[0]),i[1]+(dS_df_W0*(j[1]-i[1]))
   drw0.append(dtemp)

# find those core values in the smaller dataset that have no corresponding values
in the larger dataset
for i in rowsD0W0:
 found=0
 for j in rowsD1W0:
  if i[0] == j[0]:
   found=1
 if found==0:
  dtemp = (i[0]),i[1]+(dS_df_W0*(i[1]))
  drw0.append(dtemp)

# find those core values in the larger dataset that have no corresponding values
in the smaller dataset
for j in rowsD1W0:
 found=0
 for i in rowsD0W0:
  if i[0] == j[0]:
   found=1
 if found==0:
  if (D0W0 == 0):
   dtemp = (j[0]),(dS_df_W0*(j[1]))
  else:
   dtemp = (j[0]),((1-dS_df_W0)*(j[1]))
  drw0.append(dtemp)

#
# reduce the 2 lines drw0 and drw1 to a single line called drwr
#

# for DR assign all cores and corresponding estimated runtimes IN W1.
for i in drw0:
```

```
    for j in drw1:
     if i[0] == j[0]:
      dtemp = (i[0]),int(round(0.4999+i[1]+(dist_factor*(j[1]-i[1]))))
      drwr.append(dtemp)

   # find those core values in the smaller dataset that have no corresponding values
   in the larger dataset
   for i in drw0:
    found=0
    for j in drw1:
     if i[0] == j[0]:
      found=1
    if found==0:
     dtemp = (i[0]),int(round(0.4999+i[1]+(dist_factor*(i[1]))))
     drwr.append(dtemp)

   # find those core values in the larger dataset that have no corresponding values
   in the smaller dataset
   for j in drw1:
    found=0
    for i in drw0:
     if i[0] == j[0]:
      found=1
    if found==0:
     dtemp = (j[0]),int(round(0.4999+(dist_factor*(j[1]))))
     drwr.append(dtemp)

   # with the bm data for non matching DS calculated, update the table of
   benchmarking data
   rows=drwr

   #make a table of benchmark info that can be sorted to identify the optimum
   allocation for the non matching DS values
   for i in range(0,len(drwr)):
    bm_table.append(bmStruct(drwr[i][0]))
    bm_table[i].duration = drwr[i][1]

# sort the benchmark table on time so that we known which combination of cores gives
   the least amount of runtime
bm_table=bubble_sort(bm_table,'blah')

# use the first value in the sorted table as the "optimum" allocation
cores=bm_table[0].cores
```

```
 time=bm_table[0].duration

 # depending on cores returned from the benchmark assign nodes:ppn
 nodes=cores // 4
 if nodes == 0:
  nodes=1
  ppn=cores
 else:
  ppn = 4

 return nodes,ppn,time,rows



################################################################################
## Function to read the resources file and populate the resource_table[] based on the
    number of cores avaliable.
## The resources.txt file consists of space seperated values node number (starting
    from 0) 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([])
   # now that the node is created see how many cores it has and build up the slots
   for i in range(0,int(temp[1])):
    resource_table[int(temp[0])].append(1)
  else:
   pass

 return resource_table #FIXME(needs error handling for misformated node defmouldions)
```

```
#
    ##############################################################################################################

## Function to check if the resources required for a job are avaliable in the
    resource_table
## If sucessful returns 1, otherwise retuns 0
def find_space(nodes,ppn,jobnos):

 # Count of cpus allocated
 no_cpus = 0
 # Count of node allocated
 no_nodes = 0
 # Continuation flag to break first loop if only partial cpu requirements have been
    met.
 cont = True
 # Number of nodes which meet job requirements
 node_sum=0
 # Number of cores/per node which meet the job requirements
 core_sum=0


 logger.debug("Find Space for "+str(jobnos))
 # Traverse resource_table finding nodes where all cores are free and number of cores
    exactly matches
 # the number of cores (per node) required.
 for i in range(0,len(resource_table)):
  if len(resource_table[i]) >= ppn:
   node_sum=node_sum+1
 if node_sum>=nodes:

  # 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")
       # Exact match cannot be made, Do not continue within this loop and reset the
    cpu count
       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

# If sufficient resources have been found exit function here
if no_nodes == int(nodes):
 return 1

# Where exact resource matches cannot be found. Traverse the resource table finding
   any nodes which
# can accomodate the resources required for the job.
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]
   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

 # Check if all allocations have been made. If not return 0, else return 1
 if no_nodes != int(nodes):
  return 0
 else:
  return 1

################################################################################
## Function to assign the resources required for a job to the resource_table
## If sucessful returns an array (made_busy) of resources assigned, otherwise retuns 0
def make_busy(nodes,ppn,jobnos):
 # Count of cpus allocated
 no_cpus = 0
 # Count of nodes allocated
 no_nodes = 0
 # Continuation flag to break first loop if only partial cpu requirements have been
     met.
 cont = True
 # Array to hold values of which cores have been set to busy
 global made_busy
 made_busy = []

 logger.debug("Making Busy for "+str(jobnos))
 # Traverse resource_table finding nodes where all cores are free and number of cores
     exactly matches
 # the number of cores (per node) required.
 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
      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")
       # Exact match cannot be made, Do not continue within this loop
       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 all allocations have been made end function here
if no_nodes == int(nodes):
 print "Making Allocation: ",nodes,ppn," at: ",made_busy
 if DEBUG2:
  display_table("resource_table")
 return made_busy

# Where exact resource matches cannot be found. Traverse the resource table finding
    any nodes which
# can accomodate the resources required for the job.
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]
   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

 # Check if all allocations have been made. If not return 0, else return array of
     made_busy
 if no_nodes != int(nodes):
  return 0
 else:
  print "Making Allocation: ",nodes,ppn," at: ",made_busy
  if DEBUG2:
   display_table("resource_table")
  return made_busy

################################################################################
## Function to release resources previously marked as used in the resource_table
## Takes an array arguement (busy_cores) of cordinates previousy returned from
     make_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

################################################################################
# The display_table function takes as argument a string from "sorting_table|
     queueing_table|running_table|mould_table|resources" and
# outputs a somewhat formated table to the screen. If there are no records in the
     tables (except resources) it just says so
def display_table(table):

 if table == "sorting_table":
  if len(sorting_table) == 0:
   print "Sorting table is empty"
  else:
   print "\n-----------------------------------sorting
    ----------------------------------------"
   print tabf("Trigger",12),tabf("Job No",12),tabf("Queue",12)
   for k in range(0,len(sorting_table)):
    print tabf(str(sorting_table[k].trigger_times),12),tabf(sorting_table[k].jobnos
    ,12),tabf(sorting_table[k].run_OR_sort,12)
```

```
    print "
     ------------------------------------------------------------------------------"
elif table == "queueing_table":
 if len(queueing_table) == 0:
  print "Queued table is empty"
 else:
  print "\n------------------------------------queueing
     ------------------------------------"
  print tabf("Job No",12),tabf("ctime",12),tabf("nodes",12),tabf("ppn",12),tabf("
     Duration",12)
  for k in range(0,len(queueing_table)):
   print tabf(queueing_table[k].jobnos,12),tabf(str(queueing_table[k].ctime),12),tabf
     (str(queueing_table[k].nodes),12),tabf(str(queueing_table[k].ppn),12),tabf(str(
     queueing_table[k].duration),12)
  print "
     ------------------------------------------------------------------------------"
elif table == "resource_table":
  print "\n-------------------------------------system
     ------------------------------------"
  for k in range(0,len(resource_table)):
   print resource_table[k]
  print "
     ------------------------------------------------------------------------------"
elif table == "mould_table":
 if len(mould_table) == 0:
  print "Mouldable table is empty"
 else:
  print "\n-------------------------------------mouldtab
     ------------------------------------"
  print tabf("Job No",8),tabf("ctime",11),tabf("nodes",5),tabf("ppn",3),tabf("
     Duration",8),tabf("Start",11),tabf("End",11)
  for k in range(0,len(mould_table)):
   print tabf(mould_table[k].jobnos,8),tabf(str(mould_table[k].ctime),11),tabf(str(
     mould_table[k].nodes),5),tabf(str(mould_table[k].ppn),3),tabf(str(mould_table[k].
     duration),8),tabf(str(mould_table[k].start),11),tabf(str(mould_table[k].end),11)
  print "
     ------------------------------------------------------------------------------"
else:
 if len(running_table) == 0:
  print "Running table is empty"
 else:
  print "\n------------------------------------running
     ------------------------------------"
```

```
    print tabf("Job No",8),tabf("ctime",11),tabf("nodes",5),tabf("ppn",3),tabf("
     Duration",8),tabf("Start",11),tabf("End",11),tabf("Alloc",11)
    for k in range(0,len(running_table)):
     print tabf(running_table[k].jobnos,8),tabf(str(running_table[k].ctime),11),tabf(
     str(running_table[k].nodes),5),tabf(str(running_table[k].ppn),3),tabf(str(
     running_table[k].duration),8),tabf(str(running_table[k].start),11),tabf(str(
     running_table[k].end),11),tabf(str(running_table[k].resources),11)
    print "
     ____________________________________________________________________________"


################################################################################
## Function to move (COPY!!) job information between tables. Takes arguements source
     table, source table index, destnation table,
## resources (returned from make_busy) and a sort_flag. sort_flag is used to determine
      if the detination table is populated with
## the sortStruct and if the job should be marked as running "R" or queued "Q".
     Returns the index of the source table to which
## the job data was moved (COPIED!!)
def move_job(src_tab,src_index,dest_tab,resources,sort_flag):

 dest_rec_index = ""

 if sort_flag == 'R':
  dest_tab.append(sortStruct(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(sortStruct(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(jobStruct(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].user = src_tab[src_index].user
  dest_tab[len(dest_tab)-1].app_name = src_tab[src_index].app_name
  dest_tab[len(dest_tab)-1].workload_size = src_tab[src_index].workload_size
  dest_tab[len(dest_tab)-1].dataset_size = src_tab[src_index].dataset_size
  dest_tab[len(dest_tab)-1].queue = src_tab[src_index].queue
  dest_tab[len(dest_tab)-1].jobname = src_tab[src_index].jobname
```

```
  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
  dest_tab[len(dest_tab)-1].fjobno = src_tab[src_index].fjobno
  dest_tab[len(dest_tab)-1].bmData = src_tab[src_index].bmData
  dest_tab[len(dest_tab)-1].requested_nodes = src_tab[src_index].requested_nodes
  dest_tab[len(dest_tab)-1].requested_ppn = src_tab[src_index].requested_ppn
  dest_tab[len(dest_tab)-1].orig_duration = src_tab[src_index].orig_duration
  # If placeholder value for resources is passed set to 0
  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

################################################################################
## Function to find which indicies are populated by a particulr jobno. Takes
    arguements of job number and table to check.
## Returns index which matches job number
def find_index(jobno,table):

 for i in range(0,len(table)):
  if table[i].jobnos == jobno:
   return i

################################################################################
## Function which copies job information into the running_table. While also calling
    make busy to take resources.
## Takes arguements of index of source table (as returned by find_index), source table
     and value of previous_trigger (taken from
## previously popped job trigger_time). Uses previous_trigger to determine if start
    and end time of currently considered job needs
## to be modified. Returns index value for running_table for job which has been set to
     "running".
def make_running(i,table,previous_trigger):

 print "Allocating job  : ",table[i].jobnos,table[i].nodes,table[i].ppn
 if 0:
```

```
  display_table("resource_table")
 running_job_index=move_job(table,i,running_table,make_busy(table[i].nodes,table[i].
     ppn,table[i].jobnos),-999)

 # If previous trigger had a value other than 0 eveluate correct start and end times
     based on this value.
 if previous_trigger != 0:
  if previous_trigger<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=previous_trigger
 # Else use creation and elegible time to calculate start and end
 else:
  running_table[running_job_index].start=running_table[running_job_index].ctime



 # Udpate values
 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

 if DEBUG2:
  display_table("running_table")
  display_table("queueing_table")
  display_table("sorting_table")

 return running_job_index

##############################################################################
## Function which copies job information into the queueing_table. While also calling
     make busy to take resources.
## Takes argument of mould_table index for job being considered.
def make_queued(i):

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

```
###############################################################################
class SubOptStruct:
 def __init__(self, option):
  self.options = option
  jobs_to_pop = []
  allocation = ""
  end = 0



def get_sub_opt(bmData,ctime,oalloc,opt_end):

 global sub_opt_tab
 sub_opt_tab = []
 opt = 0


 for m in range(0,len(bmData)):
  #print "first for is ",m
  for n in range(0,len(running_table)):
  # print "second for is ",n
   if bmData[m][0] <= oalloc:
  #  print "inside first if"
  #  print "running_table[n].end + bmData[m][1] ",running_table[n].end + bmData[m][1]
  #  print "opt_end ",opt_end
  #  print (running_table[n].end + bmData[m][1]) < opt_end
    if (running_table[n].end + bmData[m][1]) < opt_end:
  #   print "inside second if"
     sub_opt_tab.append(SubOptStruct(opt))
     sub_opt_tab[opt].jobs_to_pop = running_table[n].jobnos
     sub_opt_tab[opt].allocation = bmData[m][0]
     sub_opt_tab[opt].end = running_table[n].end + bmData[m][1]
     opt += 1
  if (ctime + bmData[m][1]) < opt_end:
  # print "inside third if"
   sub_opt_tab.append(SubOptStruct(opt))
   sub_opt_tab[opt].jobs_to_pop = "0"
   sub_opt_tab[opt].allocation = bmData[m][0]
   sub_opt_tab[opt].end = ctime + bmData[m][1]
```

```
   opt += 1




# depending on cores returned from the benchmark assign nodes:ppn
popped = 1
while popped:
 popped = 0
 for p in range(0,len(sub_opt_tab)):
  cores = sub_opt_tab[p].allocation
  nodes = cores // 4
  if nodes == 0:
   nodes=1
   ppn = cores
  else:
   ppn = 4

  if not find_space(nodes,ppn,"hello") and sub_opt_tab[p].jobs_to_pop == "0":
   #print "No space. Popping"
   sub_opt_tab.pop(p)
   popped = 1
   break
  #else:
  # print "There is space for this"


bubble_sort(sub_opt_tab,"end")
if len(sub_opt_tab) != 0:
 cores = sub_opt_tab[0].allocation
 nodes = cores // 4
 if nodes == 0:
  nodes=1
  ppn = cores
 else:
  ppn = 4

 return find_space(nodes,ppn,"me!!")
else:
 return 0

#if len(sub_opt_tab) == 0:
# print "Sub Optimal table is empty"
```

```
 #else:
 # print "\n-------------------------------------Sub_Opt
    ---------------------------------------"
 # print tabf("Option",12),tabf("Pop",12),tabf("Allocation",12),tabf("End time",12)
 # for o in range(0,len(sub_opt_tab)):
 #  print tabf(str(sub_opt_tab[o].options),12),tabf(sub_opt_tab[o].jobs_to_pop,12),
    tabf(str(sub_opt_tab[o].allocation),12),tabf(str(sub_opt_tab[o].end),12)
 # print
    "-------------------------------------------------------------------------------------------------------"


################################################################################

def run_time(req_cores):

 av_cores = 0

 # Find those nodes that match exactly
 for i in range(0,len(resource_table)):
    for j in range(0,len(resource_table[i])):
     if resource_table[i][j] == 1:
      av_cores = av_cores + 1
 #print "Avaliable cores in array " , av_cores
 #print "Requested cores from system " , req_cores

 for k in range(0,len(running_table)):
  av_cores = (running_table[k].nodes*running_table[k].ppn) + av_cores
  if av_cores >= req_cores:
 #  print "Start time woud be ",running_table[k].end
   return running_table[k].end
 # print "Avaliable cores inc running " , av_cores

################################################################################
## Function to convert values in seconds to Hours:Minutes:Seconds
def format_seconds_to_hhmmss(seconds):
 hours = seconds // (60*60)
 seconds %= (60*60)
 minutes = seconds // 60
 seconds %= 60
 return "%02i:%02i:%02i" % (hours, minutes, seconds)
################################################################################
## Function to print final job description to a torque style log file.
## This can be passed the result of a pop an any table using jobStruct.
```

```
def print_logs(completed_job):

 # Convert job epoch end time to a human redable string
 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("%s;E;%s;user=%s jobname=%s queue=%s ctime=%s qtime=%s etime=%s start=%
    s end=%s Resource_List.nodes=%s:ppn=%s resources_used.walltime=%s app_name=%s
    iterations=%s dataset_size=%s\n" % (tstamp, completed_job.fjobno, completed_job.
    user, completed_job.jobname, completed_job.queue, completed_job.ctime,
    completed_job.qtime, completed_job.etime, completed_job.start, completed_job.end,
    completed_job.nodes, completed_job.ppn, format_seconds_to_hhmmss(completed_job.
    duration), completed_job.app_name, completed_job.workload_size, completed_job.
    dataset_size))
 new_log.close()

################################################################################

if __name__ == "__main__":

 get_job_info(sys.argv[1])
 bubble_sort(mould_table,'mould')

 print "\n----------------------------------------moulding
    ----------------------------------------"
 print tabf("Job No",8),tabf("User",8),tabf("ctime",11),tabf("App",10),tabf("Workload"
    ,10),tabf("Dataset",10),tabf("Nodes",5),tabf("Ppn",5),tabf("RNode",5),tabf("R_Ppn"
    ,5),tabf("Duration",10),tabf("Benchmark Data",15)
 for k in range(0,len(mould_table)):
  print tabf(mould_table[k].jobnos,8),tabf(mould_table[k].user,8),tabf(str(mould_table
    [k].ctime),11),tabf(mould_table[k].app_name,10),tabf(str(mould_table[k].
    workload_size),10),tabf(str(mould_table[k].dataset_size),10),tabf(str(mould_table[
    k].nodes),5),tabf(str(mould_table[k].ppn),5),tabf(str(mould_table[k].
    requested_nodes),5),tabf(str(mould_table[k].requested_ppn),5),tabf(str(mould_table
    [k].duration),10),tabf(str(mould_table[k].bmData),15)

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

```
for i in range(0,mould_tab_length):
 print "New Job: %s" % (mould_table[i].jobnos)
 if len(running_table) == 0 and len(queueing_table) == 0:
  print "System is totally empty - P0"
  make_running(i,mould_table,0)
  bubble_sort(running_table,"end")
 else:
  while( len(running_table) != 0 and mould_table[i].ctime > running_table[0].end):
   if mould_table[i].ctime > running_table[0].end:
    make_free(running_table[0].resources)
    completed_job=running_table.pop(0)
    print_logs(completed_job)
    print "Popped: %s - P1" % (completed_job.jobnos)

    # since we have popped lets see if we can make something run
    if len(queueing_table) != 0:
     if find_space(queueing_table[0].nodes,queueing_table[0].ppn,queueing_table[0].
   jobnos):
      print "Making a queued job run - P1a"
      make_running(0,queueing_table,completed_job.end)
      bubble_sort(running_table,"end")
      queueing_table.pop(0)
     else:
      print "No space to run queued job - P1"
    else:
     print "Nothing in the queued table - P1"
   else:
    print "Nothing to pop from running - P1"
  cantrun=1

  while cantrun and len(queueing_table) != 0:
   if find_space(queueing_table[0].nodes,queueing_table[0].ppn,queueing_table[0].
   jobnos):
    # suboptimals may come in play here FIXME
    print "Making a queued job run - P1b"
    make_running(0,queueing_table,completed_job.end)
    bubble_sort(running_table,"end")
    queueing_table.pop(0)
   else:
```

```
    print "Queued job " + queueing_table[0].jobnos + " can not run - P1b - maybe a
   suboptimal?"
    if queueing_table[0].requested_nodes == 0:
     subTrue=get_sub_opt(queueing_table[0].bmData,queueing_table[0].ctime,
   queueing_table[0].nodes*queueing_table[0].ppn,(run_time(queueing_table[0].nodes*
   queueing_table[0].ppn))+queueing_table[0].duration)
     if subTrue:
      print "I can run this suboptimally - P1b"
      cores = sub_opt_tab[0].allocation
      nodes = cores // 4
      if nodes == 0:
       nodes=1
       ppn = cores
      else:
       ppn = 4
      queueing_table[0].nodes=nodes
      queueing_table[0].ppn=ppn
      make_running(0,queueing_table,completed_job.end)
      bubble_sort(running_table,"end")
      queueing_table.pop(0)
     else:
      print "No Suboptimal Solution for this queued job -P1b"
      cantrun=0
    else:
     print "This job is unfortunately serial. - P1b"
     cantrun=0

if len(queueing_table) != 0:
 print "Stuff in the queue, gotta queue this %s - P1 - no suboptimals" % (
 queueing_table[0].jobnos)
 make_queued(i)
else:
 if find_space(mould_table[i].nodes,mould_table[i].ppn,mould_table[i].jobnos):
  print "There is space run it - P1"
  make_running(i,mould_table,0)
  bubble_sort(running_table,"end")
 else:
  print "No space gotta queue - P1 - showing suboptimals maybe they can run"
  if mould_table[i].requested_nodes == 0:
   subTrue=get_sub_opt(mould_table[i].bmData,mould_table[i].ctime,mould_table[i].
   nodes*mould_table[i].ppn,(run_time(mould_table[i].nodes*mould_table[i].ppn))+
   mould_table[i].duration)
   if subTrue:
```

```
        print "I can run this suboptimally - P1"
        cores = sub_opt_tab[0].allocation
        nodes = cores // 4
        if nodes == 0:
         nodes=1
         ppn = cores
        else:
         ppn = 4
        mould_table[i].nodes=nodes
        mould_table[i].ppn=ppn
        make_running(i,mould_table,0)
        bubble_sort(running_table,"end")
       else:
        print "Cant run this job suboptimally, lets queue for now P1"
        make_queued(i)
      else:
       print "This Job is unfortunately serial - P1"
       make_queued(i)

print "\nBegining Cleanup!\n"

while len(running_table) != 0 or len(queueing_table) != 0:
 while( len(running_table) != 0 ):
  make_free(running_table[0].resources)
  completed_job=running_table.pop(0)
  print_logs(completed_job)
  print "Popped: %s - P2" % (completed_job.jobnos)

  # since we have popped lets see if we can make something run
  cantrun=1
  while cantrun:
   if len(queueing_table) != 0:
    if find_space(queueing_table[0].nodes,queueing_table[0].ppn,queueing_table[0].
   jobnos):
     print "Making a queued job run - P2a"
     make_running(0,queueing_table,completed_job.end)
     bubble_sort(running_table,"end")
     queueing_table.pop(0)
    else:
     print "No space to run queued job - P2a - checking suboptimal"
     if queueing_table[0].requested_nodes == 0:
```

```
      subTrue=get_sub_opt(queueing_table[0].bmData,queueing_table[0].ctime,
   queueing_table[0].nodes*queueing_table[0].ppn,(run_time(queueing_table[0].nodes*
   queueing_table[0].ppn))+queueing_table[0].duration)
      if subTrue:
       print "I can run this suboptimally -P2a"
       cores = sub_opt_tab[0].allocation
       nodes = cores // 4
       if nodes == 0:
        nodes=1
        ppn = cores
       else:
        ppn = 4
       queueing_table[0].nodes=nodes
       queueing_table[0].ppn=ppn
       make_running(0,queueing_table,completed_job.end)
       bubble_sort(running_table,"end")
       queueing_table.pop(0)
      else:
       print "No Suboptimal Solution for this queued job"
       cantrun=0
     else:
      print "This Job is unfortunately serial - P2a"
      cantrun=0
  else:
   print "Nothing in the queued table - P2a"
   cantrun=0

# Fix me : is this even required?
cantrun=1
while cantrun and len(queueing_table) != 0:
 if find_space(queueing_table[0].nodes,queueing_table[0].ppn,queueing_table[0].
  jobnos):
  print "Making a queued job run - P2b"
  make_running(0,queueing_table,completed_job.end)
  bubble_sort(running_table,"end")
  queueing_table.pop(0)
 else:
  print "Queued jobs can not run -P2e- checking suboptimal"
  if queueing_table[0].requested_nodes == 0:
   subTrue=get_sub_opt(queueing_table[0].bmData,queueing_table[0].ctime,
   queueing_table[0].nodes*queueing_table[0].ppn,(run_time(queueing_table[0].nodes*
   queueing_table[0].ppn))+queueing_table[0].duration)
   if subTrue:
```

```
      print "I can run this suboptimally"
      cores = sub_opt_tab[0].allocation
      nodes = cores // 4
      if nodes == 0:
       nodes=1
       ppn = cores
      else:
       ppn = 4
      queueing_table[0].nodes=nodes
      queueing_table[0].ppn=ppn
      make_running(0,queueing_table,completed_job.end)
      bubble_sort(running_table,"end")
      queueing_table.pop(0)
     else:
      print "No Suboptimal Solution for this queued job"
      cantrun=0
    else:
     print "This job is unfortunately serial -P2b"
     cantrun=0


print "\n\n"

print "\n----------closing statement
   ----------"
print "Jobs processed: ",str(len(mould_table))
print "
   ----------"


display_table("running_table")

display_table("queueing_table")

display_table("sorting_table")

display_table("resource_table")

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

Appendix D

Appendix D: Application and System Performance Profiler Code

Below is a truncated version of the ASPP code.

D.1 Code

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

#
# Application and System Performance Profiler Toolkit
#     All Right Reserved: (c) Ibad Kureshi 2012
#
# filename: aspp.py
#
# This is the main starting script to launch the benchmarking suite. It is called
# from the base installation of the ASPP toolkit. This program starts and ensures
# all required env variables and config files are correctly configured and present.
# The program starts with an argument which refers to the family of benchmarks it
# must run. After the error checking aspp.py checks to see if has already run this
# family of benchmarks. If not it launches the pre-processor function. If it has
# already run it finds a benchmarks/<bm-family-name>.0 file and so starts the post
# processor.
```

```
#


#
# Headers
#
import sys
import os
sys.path.append('includes/')
import ofbm
import postofbm

#
# functions
#
def welcoMat():
 #
 # This function just publishes program information and is called my the main
 # function.
 #
 print '\nApplication and System Performance Profiler v0.1-1'
 print 'All Right Reserved: (c) Ibad Kureshi 2012\n'
#
# welcoMat function ends
#

def corrUsage():
 #
 # This function publishes the correct methods to invoke the aspp toolkit. It
 # is called by the main function if the main function feels that the way it
 # has been invoked is incorrect or the benchmark config file is missing
 #
 print 'You are seeing this message as ASPC was not instantiated correctly. The
    correct syntax is: '
 print '\n$ aspc <benchmark-name> \n'
 print 'You must ensure that a config file for the above benchmark is present in the
    benchmarks \nfolder. Relative to the excution folder this configuration file
    should be of the form \n./benchmarks/<benchmark-name>.conf\n'
#
# corrUsage function ends
#

def exitMat(myFLG):
 #
```

```
 # This function closes off the program and outputs a different message depending
 # on whether it was on a pre or post operation cycle. It takes as argument a
 # string that declares either pre or post.
 #
 if myFLG == 'post':
  print '\nCompleted the post processing'
  print '\nASPP has terminated with no errors!'
 elif myFLG == 'pre':
  print '\nCompleted the pre-processing. Please ensure that all jobs run through with
    no'
  print 'errors. Then run aspp again to start the post-processor.'
  print '\nASPP has terminated with no errors!'
 print '... Goodbye!\n'
#
# exitMat function ends
#


#
# main fuction
#
os.system("clear")

#
# Error handling to ensure there is always an argument and that the
# call to benchmark an application always has a local config
#
if len(sys.argv) < 2:
 welcoMat()
 corrUsage()
 exit(1)
else:
 welcoMat()
 print 'Benchmarking Application: ' + sys.argv[1] + '\n'

confFile='benchmarks/' + sys.argv[1] + '.conf'
try:
 with open(confFile) as f: pass
except IOError as e:
 print 'File ' + confFile + ' not found\n'
 corrUsage()
 exit(1)

#
```

```
# Create a workfolder for the benchmarks
#
if not os.path.exists('benchmarks/'+sys.argv[1]):
 os.makedirs('benchmarks/'+sys.argv[1])

#
# Check if this is a pre or post cycle and call appropriate functions
#
simFile='benchmarks/' + sys.argv[1] + '.0'
if (os.path.isfile(simFile)):
 print 'Found ' + simFile + ' ...'
 postofbm.postBM(sys.argv[1],confFile,simFile)
 exitMat('post')
else:
 print 'Found only ' + confFile + ' ...'
 ofbm.parseBM(confFile,sys.argv[1])
 exitMat('pre')

exit(0)

#!/usr/bin/python
# -*- coding: utf-8 -*-

#
# Application and System Performance Profiler Toolkit
#     All Right Reserved: (c) Ibad Kureshi 2012
#
# filename: ofbm.py
#
# This file is located in the includes/ folder for aspp and if called on my the
# main function. It is used by the preprocessing functions. A list of functions
# in this file follow below along with VERY short descriptions of what they do.
# Detailed descriptions can be found in the comments after each function
# declaration.
#
# Functions:
# (1) parseBM - start the preprocessor, load all configs, generate all benchmarks,
#                     submit to the system, prep for post processor
# (2) createAPPDB - create a record for the benchmark in the applications table
#                     in the aspp database.
# (3) submitSc - generate a job script for each benchmark and create an apprpriate
#                     workfolder for the particular benchmark job. submit the job
#                     script and return the job number
```

```
#


#
# Headers
#
import ConfigParser, os, subprocess as sub, math, shutil, time
import MySQLdb as mdb

#
# a class structure which will hold an array of system classifications
#
class Struct:
 def __init__(self, name):
  self.names = name
  values = None


def parseBM(fileName,BMNAME):
 #
 # This is the main pre-processor function in aspp and is called on by the main
    program
 # The main program passes the location for the benchmark configuration file and the
 # base name for the family of benchmarks being run. This script reads in the system
 # and application configuration. Decides how many workloads and dataset combinations
 # that need to be run and then assigns them cores according to the system
    classifications.
 # Then it calls on the submitSc functions to actually start the benchmarks. The main
 # function keeps a record of all the benchmarks run and their respective job numbers
    in
 # a flag file which is located in benchmarks/<bm-family-name>.0. This function then
 # calls on createAPPDB which uploads the application information to mysql.
 #


 #
 # Read in the config files for the benchmark and system config
 #
 configBm = ConfigParser.RawConfigParser()
 configClassif = ConfigParser.RawConfigParser()
 configBm.read(fileName)

 #
 # Declare variables
 #
```

```
#
# variables for the Benchmark Configuration files
#
numWL = configBm.getint('WORKLOADS', 'number-of-workloads')
metaWL = configBm.get('WORKLOADS', 'workload-meta')
numMOD = configBm.getint('MODELS', 'number-of-models')
metaMOD = configBm.get('MODELS', 'model-meta')
wLoads = []
models = []
mClass = []
bmJob = []

#
# Variables required to parse the system configuration file
#
counter = 0
systems = []

#
# variables to prepare data for the post processing phase
#
ofbmInput = open('benchmarks/'+ BMNAME +'.0','w')

#
# identify all the workloads and strip them of meta
#
print '\nFound ' + str(numWL) + ' workloads to be run.'

for x in range(0,numWL):
 wl = 'workload-' + str(x+1)
 wLoads.append(configBm.get('WORKLOADS', wl))
 print 'Workload number = ' + str(x+1) + ' is ' + wLoads[x] + ' ' + metaWL

#
# identify all datasets and strip them of meta
#
print '\nFound ' + str(numMOD) + ' models to be run.'

for x in range(0,numMOD):
 myClass = []

 mo = 'model-' + str(x+1)
```

```
 cl = 'class-' + str(x+1)

 models.append(configBm.get('MODELS', mo))
 myClass.append(configBm.get('MODELS', cl))
 bm = myClass[0]+'-fileName'
 myClass.append(configBm.get('BENCHMARK-INPUTS', bm))
 mClass.append(myClass)
 print 'Model number ' + str(x+1) + ' is ' + models[x] + ' ' + metaMOD + ' and is
    classed as ' + mClass[x][0] + ' and has input file ' + mClass[x][1]

#
# Read in the application specific configuration to run the app.
#
launchCmd = configBm.get('LAUNCHLINE', 'applaunch')

#
# Read in the system classification configuration
#
configClassif.read('config/sysinfo.conf')
numClass = configClassif.getint('CLASSIFICATIONS','number-of-classifications')

for counter in range(0,numClass):
 serStringN = 'classification-' + str(counter+1) + '-name'
 serStringV = 'classification-' + str(counter+1) + '-values'
 temp0 = configClassif.get('CLASSIFICATIONS',serStringN)
 systems.append(Struct(temp0))
 myValues = configClassif.get('CLASSIFICATIONS',serStringV)
 temp = myValues.split(',')
 systems[counter].values = []
 for i in range(len(temp)):
  systems[counter].values.append(temp[i])

#
# Begin loops to determine each benchmark job to run. This will be #workload*#dataset
    *classification[]. Classification always relates to at least 3 values
#
for x in range(0,numWL):
 for y in range(0,numMOD):
  for z in range(0,len(systems)):
   if systems[z].names == mClass[y][0]:
    for numCores in systems[z].values:
     print 'Generating ' + mClass[y][0] + ' benchmark job of ' + wLoads[x] + ' ' +
    metaWL + ' and ' + models[y] + ' ' + metaMOD + ' on ' + numCores + ' cores'
```

```
        jobid = submitSc(numCores,mClass[y][0],mClass[y][1],wLoads[x],models[y],
    launchCmd,BMNAME)
        dump = str(mClass[y][0] + ' '+ wLoads[x] + ' '+ models[y] + ' '+ numCores + ' '+
     jobid + '\n')
        ofbmInput.write(dump)

 #
 # Upload the application details and meta information to backend database.
 #
 createAPPDB(BMNAME,metaMOD,metaWL)

 #
 # Close all text files
 #
 ofbmInput.close
#
# parseBM function ends
#

def createAPPDB(BMNAME,metaMOD,metaWL):
 #
 # This function is called on by the parseBM function from the preprocessor. It is
     only
 # called per run of the pre-processor. It takes as argument the base benchmark family
 # name the meta name for the workload and the metaname for the model sizes. This
     function
 # reads the system configuration info to get all the mysql related information and
     makes
 # a connection to mysql. Then in the applications table this function added the
     benchmark
 # record which includes the family name and the two meta keywords.
 #
 configSys = ConfigParser.RawConfigParser()
 configSys.read('config/sysinfo.conf')
 mysqlHost = configSys.get('MYSQL','mysql-host')
 mysqlDB = configSys.get('MYSQL','mysql-db')
 mysqlUser = configSys.get('MYSQL','mysql-user')
 mysqlPass = configSys.get('MYSQL','mysql-password')

 con = mdb.connect(mysqlHost, mysqlUser, mysqlPass, mysqlDB);
 with con:
  cur = con.cursor()
```

```
  str = "INSERT INTO applications(appName, workMeta, datameta) VALUES ('%s', '%s', '%s
      ')" % (BMNAME,metaWL,metaMOD)
  cur.execute(str)
  print '\nCreating database entry for Benchmarks: %s' % (BMNAME)
#
# createAPPDB function ends
#

def submitSc(nCores,mClass,inputFile,wLoads,models,launchCmd,BMNAME):
 #
 # This function is called on by the parseBM function from the preprocessor. It is
     called
 # for every benchmarking job that needs to be created. This is calculated and
     controlled
 # in a for loop in the parent function. This function recieves as arguments the '
     number
 # of cores required', 'the classification of this job', 'the input file (for std in
 # purposes)', 'the workload size', 'the model size', 'the application invoking
     command',
 # 'the benchmark family name' for the current benchmark job iteration.
 # This function then reads the system config file to populate env variables related
     to the
 # underlying batch system. # It creates a workfolder for the current benchmark
     iteration
 # and naviagates to it. Depending on the type of batch it goes into an if condition.
     Here
 # it creates a jobfile for submission which is batch specific and submits the job.
     The
 # function waits to capture the job number and then returns this to the main function
     .
 #

 #
 # Read in system config file
 #
 configSysinfo = ConfigParser.RawConfigParser()
 configSysinfo.read('config/sysinfo.conf')

 #
 # get system configuration information
 #
 ppn = configSysinfo.getint('SYSTEM', 'proc-per-node')
 jobManager = configSysinfo.get('WMS', 'workload-manager')
```

```
jobQueue = configSysinfo.get('WMS', 'default-queue')

#
# Identify current folder, parent work folder and final work folder, navigate to
    parent
#
bmIdfier = BMNAME + "_" + mClass + "_" + str(wLoads) + "_" + str(models) + "_" + str(
    nCores)
oldDIR = os.getcwd()
srcFol = ('../' + BMNAME + '-' + str(wLoads) + '-' + mClass)

os.chdir('benchmarks/'+BMNAME)

#
# Copy Application Input Files Generating Final Work folders
#
shutil.copytree(srcFol,bmIdfier)

#
# Calculate Number of Nodes required
#
numNodes = int(math.ceil(float(nCores)/float(ppn)))

#
# Begin Submission Run based on Job Scheduler. navigate to final work folder
#
os.chdir(bmIdfier)

if jobManager == 'TORQUE':
 jobFile = open('jobscript.job','w')

 #
 # generate lines for the jobscript stripping place holders for dynamic elements like
     input file name and number of cores
 #
 torque_preamble = "#!/bin/bash\n#PBS -l nodes=" + str(numNodes) + ":ppn=" + str(ppn)
     + "\n#PBS -j oe\n#PBS -q " + jobQueue + "\n#PBS -N " + bmIdfier + "\n"
 torque_environ = "\ncd $PBS_O_WORKDIR\n\n"
 temp = launchCmd.replace("%CORES%",nCores)
 finalLaunch = temp.replace("%INPUTFILE%",inputFile)
 jobFile.write(torque_preamble)
 jobFile.write(torque_environ)
 jobFile.write(finalLaunch)
```

```
  jobFile.write('\n')
  jobFile.close

  #
  # jobfile does not seem to be written to disk even though its closed above. the next
  # two lines just open the file in read mode and then closes it. this seems to ensure
  # that the content is written to disk #FIXME(00001)
  #
  jobFile = open('jobscript.job','r')
  jobFile.close

  #
  # submit job and capture job number
  #
  jobDIR = os.getcwd()
  jobTEMP = jobDIR + '/jobscript.job'
  qsubPIPE = "qsub " + jobTEMP
  p = sub.Popen([qsubPIPE],stdout=sub.PIPE, shell=True)
  output = p.communicate()[0].rstrip('\n')

 #
 # return to program root and return the job number back to the preprocessor
 #
 os.chdir(oldDIR)
 return output
#
# submitSc function ends
#

#!/usr/bin/python
# -*- coding: utf-8 -*-

#
# Application and System Performance Profiler Toolkit
#     All Right Reserved: (c) Ibad Kureshi 2012
#
# filename: ofbm.py
#
# This file is located in the includes/ folder for aspp and if called on my the
# main function. It is used for the postprocessing functions. A list of functions
# in this file follow below along with VERY short descriptions of what they do.
# Detailed descriptions can be found in the comments after each function
# declaration.
```

```
#
# Functions:
# (1) postBM - start the postprocessor, load all configs, read all the executed
#                    benchmarks info, get runtimes for each benchmark and upload
#                    to mysql
# (2) dispOut - a debug function to dump to screen each benchmark job info including
#                    final run times.
# (3) findTime - go through the workload manager accounting logs to get actual and
#                    final run time for each benchmark job.
# (4) bmDBPush - upload the benchmarking information to mysql database
#


#
# Headers
#
import ConfigParser, os, sys, subprocess as sub
import MySQLdb as mdb

def postBM(baseName,confFile,fileName):
 #
 # This is the main postprocessor function in aspp and is called on by the main
    program
 # The main program passes the location for the benchmark configuration file and the
 # base name for the family of benchmarks being run along with the flag file set by
    the
 # pre processor. This system reads in the system config file and the flag file and
 # creates an array of benchmark information and results. For each entry in the array
    it
 # calls a function to find the final runtime for that job. With all the information
 # gathered its calls on another function to upload the information to mysql.
 #


 #
 # Variables
 #
 configSys = ConfigParser.RawConfigParser()
 configSys.read('config/sysinfo.conf')
 mysqlHost = configSys.get('MYSQL','mysql-host')
 mysqlDB = configSys.get('MYSQL','mysql-db')
 mysqlUser = configSys.get('MYSQL','mysql-user')
 mysqlPass = configSys.get('MYSQL','mysql-password')
 WLM = configSys.get('WMS','workload-manager')
 wmsHOME = configSys.get('WMS','wms-home')
```

```
counter = 0
bMarks = []

#
# Information Mat
#
print '\nIt would appear that the benchmark for ' + baseName + ' in ' + confFile +'
    has been run already'
print 'Starting Post Processing for ' + baseName + ' benchmarks\n'

#
# Read in the meta information of the execute benchmarks
#
simfile = file(fileName, 'r')

for line in simfile:
 if line.startswith('['):
  pass
 else:
  record = line.split()
  bMarks.append(record)
  counter = counter + 1

simfile.close

print 'Found ' + str(counter) + ' benchmarks to process.\n'

for i in range(0,counter):
 timetemp=findTime(wmsHOME, WLM, bMarks[i][4])
 bMarks[i].append(timetemp)

bmDBPush(mysqlHost,mysqlPass,mysqlUser,mysqlDB,bMarks,counter,baseName)

#
# Parsed Output Screen Display fuction. - Use only for debug purposes
#
#dispOut(bMarks,counter)
#
# postBM function ends
#

def dispOut(input,counter):
 #
```

```
 # This is a debugging function and is usually called on by postBM of the post
 # processor. It takes as argument the 2D array of benchmarking results and the
 # size of the array. Then outputs the results to screen.
 #
 for i in range(0,counter):
  print '\n==== Benchmark Number ' + str(i+1) + ' ===='
  for j in range(0,6):
   if j == 0:
    print 'Benchmark Classification: ' + input[i][j]
   elif j == 1:
    print 'Benchmark Workload: ' + input[i][j]
   elif j == 2:
    print 'Benchmark Dataset: ' + input[i][j]
   elif j == 3:
    print 'Benchmark Resource (cores): ' + input[i][j]
   elif j == 4:
    print 'Benchmark Job ID: ' + input[i][j]
   elif j == 5:
    print 'Benchmark Time: ' + str(input[i][j])
#
# dispOut function ends
#

def findTime(wmsHOME,WLM,jobID):
 #
 # This function is called on by the postBM function from the postprocessor. It is
    called
 # for every benchmarking job that was run by the preprocessor. It takes as argument
    the
 # Workload manager information and the jobID for the benchmark job that is being post
 # processed. Using that job ID it traverses the accounting information for the
    workload
 # manager and finds the final running time of that job. This is then returns the
    final
 # time after converting it to seconds. If it can not find the time it returns a 0.
 #

 bmTime = 0

 #
 # start traversal of accounting logs based on the workload manager
 #
 if WLM == 'TORQUE':
```

```
  #
  # essentially the follow bash statement needs to be run to get the accounting
    information
  # (see below). Currently the information is gathered using a system call to bash (#
    FIXME(00002))
  # Since there are three pipes these are passed seperatly through pythons popen
    mechanisms
  #
  # cat `grep -rl "JOBID" /var/spool/torque/server_priv/accounting/` | grep "E;JOBID"
    | tail -c 9
  #
  findFile = 'cat `grep -rl "' + jobID + '" ' + wmsHOME + '/server_priv/accounting/`'
  grepString = "grep 'E;" + jobID + "'"
  tailString = "tail -c 9"
  pipeFile = sub.Popen([findFile],stdout=sub.PIPE, shell=True)
  pipeGrep = sub.Popen([grepString],stdin=pipeFile.stdout,stdout=sub.PIPE, shell=True)
  p = sub.Popen([tailString],stdin=pipeGrep.stdout,stdout=sub.PIPE,stderr=sub.PIPE,
    shell=True)

  #
  # run the strings and collect the output. Stripping it of new lines and ':' to get
    hours minutes
  # and seconds. Every thing is reduced to seconds and then returned to the mains or
    if no records
  # found then 0 is returned
  #
  output = p.communicate()[0].rstrip('\n')
  ftime = output.split(':')
  if output:
   bmTime = (int(ftime[0])*3600)+(int(ftime[1])*60)+int(ftime[2])
  else:
   bmTime=0
 return bmTime
#
# findTime function ends
#

def bmDBPush(mysqlHost,mysqlPass,mysqlUser,mysqlDB,bMarks,counter,baseName):
 #
 # This function is called on by the postBM function from the postprocessor. It is
    called
 # once by the postprocessor. It takes as arguments the mysql config info. a 2d array
    of
```

```
# benchmark results, the size of the array and the family name of the benchmark. This
# function makes a connection to the database and then traverses the 2d array,
    uploading
# each row into the array to a new record in the database table.
#
con = mdb.connect(mysqlHost, mysqlUser, mysqlPass, mysqlDB);
with con:
 cur = con.cursor()
 for i in range(0,counter):
  str = "INSERT INTO benchmarks(appName, workload, dataset, cores, time) VALUES ('%s
    ', '%s', '%s', '%s', '%s')" % (baseName,bMarks[i][1],bMarks[i][2],bMarks[i][3],
   bMarks[i][5])
  cur.execute(str)
  print 'Uploading Record %d of %d.' % (i+1,counter)
#
# bmDBPush function ends
#
```

Appendix E

Appendix E: Simulated Outputs

This appendix contains the on-screen output generated by the simulator against the three datasets from A. Due to the size of the dataset, verbose output settings and high debug levels (for verification) the output currently stands at 998976 lines for each of the three runs. In the interest of space these output screens have been truncated to 200 lines each (including white space). Each section of truncated data is depicted with a '[...]'. Full outputs may be downloaded from: http://www.ibadkureshi.com/thesis/appendix-e.zip.

E.1 Real data First Come First Served (FCFS)

```
Cluster Discrete Event Simulator!
---------------------------------
Launching...
Reading algorithm ... fcfs-bf-algo
Reading system definition ... resources.txt
Reading job trace ... ../../logs/original.log
--------------------------inittab-----------------------------
```

```
Job No    ctime       nodes ppn Duration Start       End
[...]
77747.0   1364999418  1     4   54       1364999418  1364999472
77761.0   1364999649  1     4   23758    1364999649  1365023407
77778.0   1365000012  1     4   15619    1365018811  1365034430
77776.0   1364999966  1     4   19777    1365018810  1365038587
77775.0   1364999921  1     4   22490    1365017691  1365040181
77738.0   1364999320  1     4   63540    1364999320  1365062860
77768.0   1364999749  1     4   71385    1365002242  1365073627
77733.0   1364999241  1     4   75661    1364999241  1365074902
101863.0  1366640324  8     4   64808    1366640325  1366705133
101864.0  1366706619  8     4   18       1366706619  1366706637
[...]
77182.0   1364858024  2     4   10       1364858024  1364858034
77181.0   1364858011  2     4   32       1364858011  1364858043
77183.0   1364858154  2     4   28       1364858154  1364858182
77184.0   1364858161  2     4   30       1364858161  1364858191
77185.0   1364858167  2     4   27       1364858167  1364858194
77186.0   1364858172  2     4   24       1364858172  1364858196
77187.0   1364858182  2     4   33       1364858182  1364858215
77192.0   1364907453  2     4   29       1364907453  1364907482
77193.0   1364907469  2     4   26       1364907469  1364907495
77194.0   1364907481  2     4   23       1364907481  1364907504
[...]
78715.0   1365027644  3     4   31       1365036667  1365036698
79221.0   1365071243  3     4   27       1365071243  1365071270
79251.0   1365072064  3     4   31       1365072064  1365072095
79348.0   1365074292  3     4   31       1365074292  1365074323
79365.0   1365074685  3     4   27       1365074686  1365074713
```

```
79397.0  1365075452  2     4   23       1365075452  1365075475
79409.0  1365075736  3     4   128      1365075736  1365075864
79432.0  1365076375  3     4   38       1365076375  1365076413
79462.0  1365076875  2     4   97       1365076876  1365076973
79537.0  1365078333  2     4   189      1365078703  1365078892
[...]
73207.0  1366897046  2     4   169930   1366970945  1367140875
73269.0  1366898750  2     4   94531    1366970967  1367065498
73002.0  1366889417  2     4   169911   1366889417  1367059328
75252.0  1366982459  2     4   35       1367034615  1367034650
75251.0  1366982458  2     4   51       1367034545  1367034596
75250.0  1366982457  2     4   24       1367034514  1367034538
75249.0  1366982457  2     4   28       1367034457  1367034485
75248.0  1366982457  2     4   18       1367034428  1367034446
75247.0  1366982456  2     4   14       1367034403  1367034417
75246.0  1366982454  2     4   17       1367034373  1367034390
[...]
------------------------------------------------------------


------------------------------------------------------------
New Job is:  77174.0  at time  1364857972
------------------------------------------------------------
Allocating job  :  77174.0 2 4
Making Allocation:  2 4  at:  [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]


----------------------------system-------------------------------
[0, 0, 0, 0]
[0, 0, 0, 0]
[1, 1, 1, 1]
```

```
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
---------------------------------------------------------------


-------------------------running-------------------------------
Job No   ctime       nodes ppn Duration Start       End         Alloc
77174.0  1364857972  2     4   114053   1364857972  1364972025
                              [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]
---------------------------------------------------------------
Queued table is empty
Sorting table is empty


---------------------------------------------------------------
New Job is:  77175.0  at time  1364857977
---------------------------------------------------------------
Allocating job  :  77175.0 2 4
Making Allocation:  2 4  at:  [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3]
```

```
--------------------------system--------------------------------
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
------------------------------------------------------------------


--------------------------------running---------------------------
Job No   ctime       nodes ppn Duration Start       End         Alloc
77174.0  1364857972  2     4   114053   1364857972  1364972025
                              [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]
77175.0  1364857977  2     4   117187   1364857977  1364975164
                              [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3]
------------------------------------------------------------------
Queued table is empty
```

```
------------------------------sorting---------------------------
Trigger      Job No       Queue
1364972025   77174.0      R
----------------------------------------------------------------


----------------------------------------------------------------
New Job is:  77176.0  at time  1364857985
----------------------------------------------------------------
Allocating job  :  77176.0 2 4
Making Allocation:  2 4  at:  [4, 0, 4, 1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]


------------------------------system----------------------------
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
```

```
[1, 1, 1, 1]
------------------------------------------------------------------


---------------------------running--------------------------------
Job No   ctime       nodes ppn Duration Start       End         Alloc
77174.0  1364857972  2     4   114053   1364857972  1364972025
                             [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]
77175.0  1364857977  2     4   117187   1364857977  1364975164
                             [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3]
77176.0  1364857984  2     4   115260   1364857985  1364973245
                             [4, 0, 4, 1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
------------------------------------------------------------------
Queued table is empty


-----------------------------sorting------------------------------
Trigger      Job No       Queue
1364972025   77174.0      R
1364975164   77175.0      R
------------------------------------------------------------------


------------------------------------------------------------------
New Job is:  77177.0  at time  1364857991
------------------------------------------------------------------
Allocating job  :  77177.0 2 4
Making Allocation:  2 4  at:  [6, 0, 6, 1, 6, 2, 6, 3, 7, 0, 7, 1, 7, 2, 7, 3]


-----------------------------system-------------------------------
[0, 0, 0, 0]
[0, 0, 0, 0]
```

```
[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

----------------------------------------------------------------


----------------------------running-------------------------------
Job No   ctime       nodes ppn Duration Start       End         Alloc
77174.0  1364857972  2     4   114053   1364857972  1364972025
                             [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]
77175.0  1364857977  2     4   117187   1364857977  1364975164
                             [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3]
77176.0  1364857984  2     4   115260   1364857985  1364973245
                             [4, 0, 4, 1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
77177.0  1364857991  2     4   112162   1364857991  1364970153
                             [6, 0, 6, 1, 6, 2, 6, 3, 7, 0, 7, 1, 7, 2, 7, 3]
----------------------------------------------------------------
Queued table is empty
```

```
----------------------------sorting-------------------------------
Trigger      Job No       Queue
1364972025   77174.0      R
1364973245   77176.0      R
1364975164   77175.0      R
------------------------------------------------------------------


------------------------------------------------------------------
New Job is:  77178.0  at time  1364857996
------------------------------------------------------------------
Allocating job  :  77178.0 2 4
Making Allocation:  2 4  at:  [8, 0, 8, 1, 8, 2, 8, 3, 9, 0, 9, 1, 9, 2, 9, 3]
[...]
Calling inside while (poped jobs)


----------------------------sorting-------------------------------
Trigger      Job No       Queue
1366707149   101867.0     Q
1367855347   101863.0     R
------------------------------------------------------------------
Setting a Queued job with index:  0  and jobid:  101867.0  to running


Allocating job  :  101867.0 8 4
Making Allocation:  8 4  at:  [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3, 6


----------------------------system--------------------------------
[0, 0, 0, 0]
[0, 0, 0, 0]
```

```
[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[1, 1, 1, 1]

----------------------------------------------------------------


------------------------------running-------------------------------

Job No    ctime       nodes ppn Duration Start       End         Alloc

101863.0 1366640324  8     4   64808    1367790539  1367855347

                     [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3, 4, 0, 4,

                     1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3, 12, 0, 12, 1, 12,

                     2, 12, 3, 13, 0, 13, 1, 13, 2, 13, 3, 14, 0, 14, 1, 14, 2

                     , 14, 3, 15, 0, 15, 1, 15, 2, 15, 3]

101867.0 1366707149  8     4   72166    1367824858  1367897024

                     [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3, 6, 0, 6,

                     1, 6, 2, 6, 3, 7, 0, 7, 1, 7, 2, 7, 3, 8, 0, 8, 1, 8, 2,

                     8, 3, 9, 0, 9, 1, 9, 2, 9, 3, 10, 0, 10, 1, 10, 2, 10, 3,

                     11, 0, 11, 1, 11, 2, 11, 3]
```

```
------------------------------------------------------------------


------------------------------queueing--------------------------
Job No       ctime        nodes        ppn          Duration
101867.0     1366707149   8            4            72166
------------------------------------------------------------------


----------------------------sorting-------------------------------
Trigger      Job No       Queue
1366707149   101867.0     Q
1367855347   101863.0     R
------------------------------------------------------------------
Popping:  101863.0  from  [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3, 4,
0, 4, 1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3, 12, 0, 12, 1, 12, 2, 12, 3, 13, 0,
13, 1, 13, 2, 13, 3, 14, 0, 14, 1, 14, 2, 14, 3, 15, 0, 15, 1, 15, 2, 15, 3]
------------------------------------------------------------------
Calling inside while (poped jobs)


----------------------------sorting-------------------------------
Trigger      Job No       Queue
1367897024   101867.0     R
------------------------------------------------------------------
Popping:  101867.0  from  [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3, 6,
0, 6, 1, 6, 2, 6, 3, 7, 0, 7, 1, 7, 2, 7, 3, 8, 0, 8, 1, 8, 2, 8, 3, 9, 0, 9,
1, 9, 2, 9, 3, 10, 0, 10, 1, 10, 2, 10, 3, 11, 0, 11, 1, 11, 2, 11, 3]
------------------------------------------------------------------
Calling inside while (poped jobs)
Sorting table is empty
```

```
--------------------closing statement---------------------------

Jobs processed:  1150

----------------------------------------------------------------

Running table is empty

Queued table is empty

Sorting table is empty


--------------------------------system----------------------------

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

----------------------------------------------------------------
```

E.2 Normalised data with FCFS

```
Cluster Discrete Event Simulator!
---------------------------------
Launching...
Reading algorithm ... fcfs-bf-algo
Reading system definition ... resources.txt
Reading job trace ... ../../logs/normalised_moulded.log
-------------------------inittab-------------------------------
Job No    ctime       nodes ppn Duration Start       End
73207.0   1366897046  2     4   128698   1366970945  1367099643
73269.0   1366898750  2     4   128698   1366970967  1367099665
73002.0   1366889417  2     4   128698   1366889417  1367018115
75252.0   1366982459  2     4   43       1367034615  1367034658
75251.0   1366982458  2     4   43       1367034545  1367034588
75250.0   1366982457  2     4   43       1367034514  1367034557
75249.0   1366982457  2     4   43       1367034457  1367034500
75248.0   1366982457  2     4   43       1367034428  1367034471
75247.0   1366982456  2     4   43       1367034403  1367034446
75246.0   1366982454  2     4   43       1367034373  1367034416
[...]
77747.0   1364999418  1     4   82850    1364999418  1365082268
77761.0   1364999649  1     4   82850    1364999649  1365082499
77778.0   1365000012  1     4   82850    1365018811  1365101661
77776.0   1364999966  1     4   82850    1365018810  1365101660
77775.0   1364999921  1     4   82850    1365017691  1365100541
77738.0   1364999320  1     4   63540    1364999320  1365062860
77768.0   1364999749  1     4   71385    1365002242  1365073627
77733.0   1364999241  1     4   75661    1364999241  1365074902
```

```
101863.0 1366640324  8     4    161256    1366640325  1366801581
101864.0 1366706619  8     4    17        1366706619  1366706636
[...]
77182.0  1364858024  2     4    26        1364858024  1364858050
77181.0  1364858011  2     4    26        1364858011  1364858037
77183.0  1364858154  2     4    26        1364858154  1364858180
77184.0  1364858161  2     4    26        1364858161  1364858187
77185.0  1364858167  2     4    26        1364858167  1364858193
77186.0  1364858172  2     4    26        1364858172  1364858198
77187.0  1364858182  2     4    26        1364858182  1364858208
77192.0  1364907453  2     4    26        1364907453  1364907479
77193.0  1364907469  2     4    26        1364907469  1364907495
77194.0  1364907481  2     4    26        1364907481  1364907507
[...]
78715.0  1365027644  3     4    156       1365036667  1365036823
79221.0  1365071243  3     4    156       1365071243  1365071399
79251.0  1365072064  3     4    156       1365072064  1365072220
79348.0  1365074292  3     4    156       1365074292  1365074448
79365.0  1365074685  3     4    156       1365074686  1365074842
79397.0  1365075452  2     4    156       1365075452  1365075608
79409.0  1365075736  3     4    156       1365075736  1365075892
79432.0  1365076375  3     4    156       1365076375  1365076531
79462.0  1365076875  2     4    156       1365076876  1365077032
79537.0  1365078333  2     4    156       1365078703  1365078859
[...]
-------------------------------------------------------------


-------------------------------------------------------------
New Job is:  77174.0  at time  1364857972
```

```
-------------------------------------------------------------

Allocating job  :  77174.0 2 4

Making Allocation:  2 4  at:  [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]


---------------------------system----------------------------

[0, 0, 0, 0]

[0, 0, 0, 0]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]
---------------------------------------------------------------


------------------------------running-------------------------

Job No   ctime       nodes ppn Duration Start       End         Alloc

77174.0  1364857972  2     4    116864   1364857972 1364974836

                              [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]

---------------------------------------------------------------
```

```
Queued table is empty

Sorting table is empty


----------------------------------------------------------------

New Job is:  77175.0  at time  1364857977

----------------------------------------------------------------

Allocating job  :  77175.0 2 4

Making Allocation:  2 4  at:  [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3]


----------------------------system--------------------------------

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

----------------------------------------------------------------
```

```
---------------------------running--------------------------------
Job No    ctime       nodes ppn Duration Start       End         Alloc
77174.0   1364857972  2     4   116864   1364857972  1364974836
                                [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]
77175.0   1364857977  2     4   116864   1364857977  1364974841
                                [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3]
-------------------------------------------------------------------
Queued table is empty


--------------------------sorting----------------------------------
Trigger      Job No       Queue
1364974836   77174.0      R
-------------------------------------------------------------------


-------------------------------------------------------------------
New Job is:  77176.0  at time  1364857985
-------------------------------------------------------------------
Allocating job  :  77176.0 2 4
Making Allocation:  2 4  at:  [4, 0, 4, 1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]


----------------------------system---------------------------------
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[1, 1, 1, 1]
[1, 1, 1, 1]
```

```
[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

----------------------------------------------------------------


----------------------------running-------------------------------
Job No   ctime       nodes ppn Duration Start       End         Alloc
77174.0  1364857972  2     4   116864   1364857972  1364974836
                              [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]
77175.0  1364857977  2     4   116864   1364857977  1364974841
                              [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3]
77176.0  1364857984  2     4   116864   1364857985  1364974849
                              [4, 0, 4, 1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
----------------------------------------------------------------
Queued table is empty


-------------------------sorting----------------------------------
Trigger      Job No       Queue
1364974836   77174.0      R
1364974841   77175.0      R
----------------------------------------------------------------


----------------------------------------------------------------
```

```
New Job is:  77177.0  at time  1364857991

---------------------------------------------------------------

Allocating job  :  77177.0 2 4

Making Allocation:  2 4  at:  [6, 0, 6, 1, 6, 2, 6, 3, 7, 0, 7, 1, 7, 2, 7, 3]


--------------------------------system-------------------------------

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

---------------------------------------------------------------


--------------------------------running----------------------------

Job No    ctime       nodes ppn Duration Start       End         Alloc

77174.0   1364857972  2     4   116864   1364857972  1364974836

                                [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3]
```

```
77175.0  1364857977  2     4   116864   1364857977  1364974841
                              [2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3]
77176.0  1364857984  2     4   116864   1364857985  1364974849
                              [4, 0, 4, 1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
77177.0  1364857991  2     4   116864   1364857991  1364974855
                              [6, 0, 6, 1, 6, 2, 6, 3, 7, 0, 7, 1, 7, 2, 7, 3]
------------------------------------------------------------------

Queued table is empty


-------------------------------sorting----------------------------
Trigger      Job No       Queue
1364974836   77174.0      R
1364974841   77175.0      R
1364974849   77176.0      R
------------------------------------------------------------------


------------------------------------------------------------------
New Job is:  77178.0  at time  1364857996
------------------------------------------------------------------
Allocating job  :  77178.0 2 4
Making Allocation:  2 4  at:  [8, 0, 8, 1, 8, 2, 8, 3, 9, 0, 9, 1, 9, 2, 9, 3]
[...]
--------------------------------system----------------------------
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
```

```
[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[1, 1, 1, 1]

----------------------------------------------------------------


----------------------------running---------------------------------
Job No    ctime       nodes ppn Duration Start       End         Alloc
101863.0 1366640324  8     4   161256   1367793937  1367955193
                                 [7, 0, 7, 1, 7, 2, 7, 3, 8, 0, 8, 1, 8, 2, 8, 3
                                 , 9, 0, 9, 1, 9, 2, 9, 3, 10, 0, 10, 1, 10, 2,
                                 10, 3, 11, 0, 11, 1, 11, 2, 11, 3, 12, 0, 12, 1
                                 , 12, 2, 12, 3, 13, 0, 13, 1, 13, 2, 13, 3, 15,
                                 0, 15, 1, 15, 2, 15, 3]
101867.0 1366707149  8     4   161256   1367826862  1367988118
                                 [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3
                                 , 2, 0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3,
                                 3, 4, 0, 4, 1, 4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5
                                 , 3, 6, 0, 6, 1, 6, 2, 6, 3, 14, 0, 14, 1, 14,
                                 2, 14, 3]
----------------------------------------------------------------
```

```
----------------------------queueing-------------------------------
Job No       ctime        nodes        ppn          Duration
101867.0     1366707149   8            4            161256
-------------------------------------------------------------------


----------------------------sorting--------------------------------
Trigger      Job No       Queue
1366707149   101867.0     Q
1367955193   101863.0     R
-------------------------------------------------------------------
Popping:  101863.0  from  [7, 0, 7, 1, 7, 2, 7, 3, 8, 0, 8, 1, 8, 2, 8, 3, 9,
0, 9, 1, 9, 2, 9, 3, 10, 0, 10, 1, 10, 2, 10, 3, 11, 0, 11, 1, 11, 2, 11, 3,
12, 0, 12, 1, 12, 2, 12, 3, 13, 0, 13, 1, 13, 2, 13, 3, 15, 0, 15, 1, 15, 2,
 15, 3]
-------------------------------------------------------------------
Calling inside while (poped jobs)


----------------------------sorting--------------------------------
Trigger      Job No       Queue
1367988118   101867.0     R
-------------------------------------------------------------------
Popping:  101867.0  from  [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1, 2, 1, 3, 2,
0, 2, 1, 2, 2, 2, 3, 3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1, 4, 2, 4, 3, 5, 0, 5,
1, 5, 2, 5, 3, 6, 0, 6, 1, 6, 2, 6, 3, 14, 0, 14, 1, 14, 2, 14, 3]
-------------------------------------------------------------------
Calling inside while (poped jobs)
Sorting table is empty


-------------------------closing statement-------------------------
```

```
Jobs processed:  1150

-----------------------------------------------------------------

Running table is empty

Queued table is empty

Sorting table is empty


---------------------------------system-----------------------------

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

-----------------------------------------------------------------------
```

E.3 Normalised data with Moulding

```
Cluster Discrete Event Simulator!
---------------------------------
Launching...
Reading algorithm ... mould-algo
Reading system definition ... resources.txt
Reading job trace ... ../../logs/normalised_moulded.log
---------------------------moulding-------------------------------------
Job No   User     ctime       App        Workload   Dataset    Nodes Ppn
RNode R_Ppn Duration   Benchmark Data
73207.0  mduser1 1366897046  dlpoly     900000     4000       4     4
0     0     72566      [(12L, 90018), (8L, 128698), (4L, 592012), (16L, 72566)]
73269.0  mduser1 1366898750  dlpoly     900000     4000       4     4
0     0     72566      [(12L, 90018), (8L, 128698), (4L, 592012), (16L, 72566)]
73002.0  mduser1 1366889417  dlpoly     900000     4000       4     4
0     0     72566      [(12L, 90018), (8L, 128698), (4L, 592012), (16L, 72566)]
75252.0  mduser1 1366982459  dlpoly     300        4000       4     4
0     0     25         [(12L, 31), (8L, 43), (4L, 198), (16L, 25)]
75251.0  mduser1 1366982458  dlpoly     300        4000       4     4
0     0     25         [(12L, 31), (8L, 43), (4L, 198), (16L, 25)]
75250.0  mduser1 1366982457  dlpoly     300        4000       4     4
0     0     25         [(12L, 31), (8L, 43), (4L, 198), (16L, 25)]
75249.0  mduser1 1366982457  dlpoly     300        4000       4     4
0     0     25         [(12L, 31), (8L, 43), (4L, 198), (16L, 25)]
75248.0  mduser1 1366982457  dlpoly     300        4000       4     4
0     0     25         [(12L, 31), (8L, 43), (4L, 198), (16L, 25)]
75247.0  mduser1 1366982456  dlpoly     300        4000       4     4
0     0     25         [(12L, 31), (8L, 43), (4L, 198), (16L, 25)]
```

```
75246.0  mduser1 1366982454  dlpoly     300        4000       4     4
0     0     25         [(12L, 31), (8L, 43), (4L, 198), (16L, 25)]
[...]
77747.0  mduser2 1364999418  dlpoly     5000000    500        1     4
0     0     82850      [(6L, 229075), (4L, 82850), (2L, 145510)]
77761.0  mduser2 1364999649  dlpoly     5000000    500        1     4
0     0     82850      [(6L, 229075), (4L, 82850), (2L, 145510)]
77775.0  mduser2 1364999921  dlpoly     5000000    500        1     4
0     0     82850      [(6L, 229075), (4L, 82850), (2L, 145510)]
77776.0  mduser2 1364999966  dlpoly     5000000    500        1     4
0     0     82850      [(6L, 229075), (4L, 82850), (2L, 145510)]
77778.0  mduser2 1365000012  dlpoly     5000000    500        1     4
0     0     82850      [(6L, 229075), (4L, 82850), (2L, 145510)]
77738.0  mduser2 1364999320  dlpoly     5000000    500        1     4
0     0     82850      [(6L, 229075), (4L, 82850), (2L, 145510)]
77768.0  mduser2 1364999749  dlpoly     5000000    500        1     4
0     0     82850      [(6L, 229075), (4L, 82850), (2L, 145510)]
77733.0  mduser2 1364999241  dlpoly     5000000    500        1     4
0     0     82850      [(6L, 229075), (4L, 82850), (2L, 145510)]
101863.0 mduser2 1366640324  dlpoly     1000000    8000       4     4
0     0     161256     [(12L, 200038), (8L, 285994), (4L, 1315582), (16L, 161256
101864.0 mduser2 1366706619  dlpoly     100        8000       4     4
0     0     17         [(12L, 21), (8L, 29), (4L, 132), (16L, 17)]
[...]
77181.0  cfduser1 1364858011  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77182.0  cfduser1 1364858024  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77183.0  cfduser1 1364858154  fluent     1          16000000   3     4
```

```
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77184.0  cfduser1 1364858161  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77185.0  cfduser1 1364858167  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77186.0  cfduser1 1364858172  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77187.0  cfduser1 1364858182  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77192.0  cfduser1 1364907453  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77193.0  cfduser1 1364907469  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
77194.0  cfduser1 1364907481  fluent     1          16000000   3     4
0     0     1          [(12L, 17), (8L, 26), (16L, 23), (14L, 1)]
[...]
78715.0  cfduser2 1365027644  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79221.0  cfduser2 1365071243  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79251.0  cfduser2 1365072064  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79348.0  cfduser2 1365074292  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79365.0  cfduser2 1365074685  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79397.0  cfduser2 1365075452  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79409.0  cfduser2 1365075736  fluent     12         8100000    3     4
```

```
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79432.0  cfduser2 1365076375  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79462.0  cfduser2 1365076875  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
79537.0  cfduser2 1365078333  fluent     12         8100000    3     4
0     0     98         [(8L, 156), (12L, 98), (16L, 136), (4L, 302)]
----------------------------system-------------------------------------
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
------------------------------------------------------------------------
New Job: 77174.0
System is totally empty - P0
Allocating job  :  77174.0 3 4
```

```
Making Allocation:  3 4  at:  [0, 0, 0, 1, 0, 2, 0, 3, 1, 0, 1, 1, 1,
2, 1, 3, 2, 0, 2, 1, 2, 2, 2, 3]
New Job: 77175.0
There is space run it - P1
Allocating job  :  77175.0 3 4
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1, 4,
2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
New Job: 77176.0
There is space run it - P1
Allocating job  :  77176.0 3 4
Making Allocation:  3 4  at:  [6, 0, 6, 1, 6, 2, 6, 3, 7, 0, 7, 1, 7,
2, 7, 3, 8, 0, 8, 1, 8, 2, 8, 3]
New Job: 77177.0
There is space run it - P1
Allocating job  :  77177.0 3 4
Making Allocation:  3 4  at:  [9, 0, 9, 1, 9, 2, 9, 3, 10, 0, 10, 1,
10, 2, 10, 3, 11, 0, 11, 1, 11, 2, 11, 3]
New Job: 77178.0
There is space run it - P1
Allocating job  :  77178.0 3 4
Making Allocation:  3 4  at:  [12, 0, 12, 1, 12, 2, 12, 3, 13, 0, 13,
1, 13, 2, 13, 3, 14, 0, 14, 1, 14, 2, 14, 3]
New Job: 77179.0
No space gotta queue - P1 - showing suboptimals maybe they can run
Cant run this job suboptimally, lets queue for now P1
Can not make allocation for : 77179.0 3 4
Job being queued:  77179.0
New Job: 77180.0
Queued job 77179.0 can not run - P1b - maybe a suboptimal?
```

```
No Suboptimal Solution for this queued job -P1b
Stuff in the queue, gotta queue this 77179.0 - P1 - no suboptimals
Can not make allocation for : 77180.0 3 4
Job being queued:  77180.0
New Job: 77181.0
Queued job 77179.0 can not run - P1b - maybe a suboptimal?
No Suboptimal Solution for this queued job -P1b
Stuff in the queue, gotta queue this 77179.0 - P1 - no suboptimals
Can not make allocation for : 77181.0 3 4
Job being queued:  77181.0
New Job: 77182.0
Queued job 77179.0 can not run - P1b - maybe a suboptimal?
No Suboptimal Solution for this queued job -P1b
Stuff in the queue, gotta queue this 77179.0 - P1 - no suboptimals
Can not make allocation for : 77182.0 3 4
Job being queued:  77182.0
New Job: 77183.0
Queued job 77179.0 can not run - P1b - maybe a suboptimal?
No Suboptimal Solution for this queued job -P1b
Stuff in the queue, gotta queue this 77179.0 - P1 - no suboptimals
Can not make allocation for : 77183.0 3 4
Job being queued:  77183.0
New Job: 77184.0
Queued job 77179.0 can not run - P1b - maybe a suboptimal?
No Suboptimal Solution for this queued job -P1b
Stuff in the queue, gotta queue this 77179.0 - P1 - no suboptimals
Can not make allocation for : 77184.0 3 4
Job being queued:  77184.0
[...]
```

```
Queued job 79348.0 can not run - P1b - maybe a suboptimal?
I can run this suboptimally - P1b
Allocating job  :  79348.0 1 4
Making Allocation:  1 4  at:  [6, 0, 6, 1, 6, 2, 6, 3]
Queued job 79365.0 can not run - P1b - maybe a suboptimal?
No Suboptimal Solution for this queued job -P1b
Stuff in the queue, gotta queue this 79365.0 - P1 - no suboptimals
Can not make allocation for : 79695.0 3 4
Job being queued:  79695.0
[...]
New Job: 102006.0
Popped: 102005.0 - P1
Nothing in the queued table - P1
There is space run it - P1
Allocating job  :  102006.0 3 4
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1,
4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
New Job: 102008.0
Popped: 102006.0 - P1
Nothing in the queued table - P1
There is space run it - P1
Allocating job  :  102008.0 3 4
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1,
4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
New Job: 102010.0
Popped: 102008.0 - P1
Nothing in the queued table - P1
There is space run it - P1
Allocating job  :  102010.0 3 4
```

```
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1,
4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
New Job: 102012.0
Popped: 102010.0 - P1
Nothing in the queued table - P1
There is space run it - P1
Allocating job  :  102012.0 3 4
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1,
4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
New Job: 102013.0
Popped: 102012.0 - P1
Nothing in the queued table - P1
There is space run it - P1
Allocating job  :  102013.0 3 4
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1,
4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
New Job: 102015.0
Popped: 102013.0 - P1
Nothing in the queued table - P1
There is space run it - P1
Allocating job  :  102015.0 3 4
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1,
4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
New Job: 102017.0
Popped: 102015.0 - P1
Nothing in the queued table - P1
There is space run it - P1
Allocating job  :  102017.0 3 4
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1,
```

```
4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]
New Job: 102019.0
Popped: 102017.0 - P1
Nothing in the queued table - P1
There is space run it - P1
Allocating job  :  102019.0 3 4
Making Allocation:  3 4  at:  [3, 0, 3, 1, 3, 2, 3, 3, 4, 0, 4, 1,
4, 2, 4, 3, 5, 0, 5, 1, 5, 2, 5, 3]

Begining Cleanup!

Popped: 102019.0 - P2
Nothing in the queued table - P2a
Popped: 102000.0 - P2
Nothing in the queued table - P2a

--------------------------closing statement-----------------------
Jobs processed:  1150
------------------------------------------------------------------
Running table is empty
Queued table is empty
Sorting table is empty

----------------------------system--------------------------------
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
```

```
[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 1, 1, 1]
```

Appendix F

Appendix F: Simulated Logs

This appendix contains the logs generated by the simulator against the three datasets from A. In the interest of space these logs have been truncated to only include the 40 records found in A. Full outputs may be downloaded from: http://www.ibadkureshi.com/thesis/appendix-f.zip.

F.1 Real data First Come First Served (FCFS)

```
26/02/2013 14:24:13;E;75246.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361888454 qtime=1361888636 etime=1361888454 start=1361888636 end=1361888653
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:17
26/02/2013 14:24:19;E;75247.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361888456 qtime=1361888645 etime=1361888456 start=1361888645 end=1361888659
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:14
26/02/2013 14:24:26;E;75248.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361888457 qtime=1361888648 etime=1361888457 start=1361888648 end=1361888666
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:18
26/02/2013 14:24:37;E;75250.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361888457 qtime=1361888653 etime=1361888457 start=1361888653 end=1361888677
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:24
26/02/2013 14:24:39;E;75249.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361888457 qtime=1361888651 etime=1361888457 start=1361888651 end=1361888679
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:28
```

```
26/02/2013 14:25:01;E;75252.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361888459 qtime=1361888666 etime=1361888459 start=1361888666 end=1361888701
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:35
26/02/2013 14:25:10;E;75251.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361888458 qtime=1361888659 etime=1361888458 start=1361888659 end=1361888710
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:51
26/02/2013 17:21:21;E;73269.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361804750 qtime=1361804750 etime=1361804750 start=1361804750 end=1361899281
    Resource_List.nodes=2:ppn=4 resources_used.walltime=26:15:31
27/02/2013 11:42:08;E;73002.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361795417 qtime=1361795417 etime=1361795417 start=1361795417 end=1361965328
    Resource_List.nodes=2:ppn=4 resources_used.walltime=47:11:51
27/02/2013 13:49:36;E;73207.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1361803046 qtime=1361803046 etime=1361803046 start=1361803046 end=1361972976
    Resource_List.nodes=2:ppn=4 resources_used.walltime=47:12:10
02/04/2013 00:14:03;E;77181.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858011 qtime=1364858011 etime=1364858011 start=1364858011 end=1364858043
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:32
02/04/2013 00:14:13;E;77182.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858024 qtime=1364858043 etime=1364858024 start=1364858043 end=1364858053
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:10
02/04/2013 00:16:22;E;77183.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858154 qtime=1364858154 etime=1364858154 start=1364858154 end=1364858182
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:28
02/04/2013 00:16:52;E;77184.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858161 qtime=1364858182 etime=1364858161 start=1364858182 end=1364858212
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:30
02/04/2013 00:17:19;E;77185.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858167 qtime=1364858212 etime=1364858167 start=1364858212 end=1364858239
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:27
02/04/2013 00:17:43;E;77186.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858172 qtime=1364858239 etime=1364858172 start=1364858239 end=1364858263
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:24
02/04/2013 00:18:16;E;77187.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858182 qtime=1364858263 etime=1364858182 start=1364858263 end=1364858296
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:33
03/04/2013 07:23:02;E;77192.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907453 qtime=1364970153 etime=1364907453 start=1364970153 end=1364970182
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:29
03/04/2013 07:23:28;E;77193.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907469 qtime=1364970182 etime=1364907469 start=1364970182 end=1364970208
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
```

```
03/04/2013 07:23:51;E;77194.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907481 qtime=1364970208 etime=1364907481 start=1364970208 end=1364970231
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:23
04/04/2013 17:25:28;E;77747.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999418 qtime=1365092674 etime=1364999418 start=1365092674 end=1365092728
    Resource_List.nodes=1:ppn=4 resources_used.walltime=00:00:54
04/04/2013 23:10:27;E;77778.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1365000012 qtime=1365097808 etime=1365000012 start=1365097808 end=1365113427
    Resource_List.nodes=1:ppn=4 resources_used.walltime=04:20:19
05/04/2013 00:19:45;E;77776.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999966 qtime=1365097808 etime=1364999966 start=1365097808 end=1365117585
    Resource_List.nodes=1:ppn=4 resources_used.walltime=05:29:37
05/04/2013 00:50:14;E;77761.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999649 qtime=1365095656 etime=1364999649 start=1365095656 end=1365119414
    Resource_List.nodes=1:ppn=4 resources_used.walltime=06:35:58
05/04/2013 00:53:06;E;77775.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999921 qtime=1365097096 etime=1364999921 start=1365097096 end=1365119586
    Resource_List.nodes=1:ppn=4 resources_used.walltime=06:14:50
05/04/2013 02:03:15;E;79397.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365075452 qtime=1365123772 etime=1365075452 start=1365123772 end=1365123795
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:23
05/04/2013 02:05:22;E;79462.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365076875 qtime=1365123825 etime=1365076876 start=1365123825 end=1365123922
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:01:37
05/04/2013 02:08:31;E;79537.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365078333 qtime=1365123922 etime=1365078333 start=1365123922 end=1365124111
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:03:09
05/04/2013 07:44:28;E;77738.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999320 qtime=1365080728 etime=1364999320 start=1365080728 end=1365144268
    Resource_List.nodes=1:ppn=4 resources_used.walltime=17:39:00
05/04/2013 07:44:59;E;78715.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365027644 qtime=1365144268 etime=1365027644 start=1365144268 end=1365144299
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:31
05/04/2013 07:45:26;E;79221.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365071243 qtime=1365144299 etime=1365071243 start=1365144299 end=1365144326
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:27
05/04/2013 07:45:57;E;79251.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365072064 qtime=1365144326 etime=1365072064 start=1365144326 end=1365144357
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:31
05/04/2013 07:46:28;E;79348.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365074292 qtime=1365144357 etime=1365074292 start=1365144357 end=1365144388
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:31
```

```
05/04/2013 07:46:55;E;79365.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365074685 qtime=1365144388 etime=1365074685 start=1365144388 end=1365144415
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:27
05/04/2013 07:49:03;E;79409.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365075736 qtime=1365144415 etime=1365075736 start=1365144415 end=1365144543
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:08
05/04/2013 07:49:41;E;79432.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365076375 qtime=1365144543 etime=1365076375 start=1365144543 end=1365144581
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:38
05/04/2013 10:49:30;E;77733.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999241 qtime=1365079709 etime=1364999241 start=1365079709 end=1365155370
    Resource_List.nodes=1:ppn=4 resources_used.walltime=21:01:01
05/04/2013 14:05:26;E;77768.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999749 qtime=1365095741 etime=1364999749 start=1365095741 end=1365167126
    Resource_List.nodes=1:ppn=4 resources_used.walltime=19:49:45
02/04/2013 00:14:03;E;77181.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858011 qtime=1364858011 etime=1364858011 start=1364858011 end=1364858043
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:32
02/04/2013 00:14:13;E;77182.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858024 qtime=1364858043 etime=1364858024 start=1364858043 end=1364858053
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:10
02/04/2013 00:16:22;E;77183.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858154 qtime=1364858154 etime=1364858154 start=1364858154 end=1364858182
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:28
02/04/2013 00:16:52;E;77184.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858161 qtime=1364858182 etime=1364858161 start=1364858182 end=1364858212
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:30
02/04/2013 00:17:19;E;77185.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858167 qtime=1364858212 etime=1364858167 start=1364858212 end=1364858239
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:27
02/04/2013 00:17:43;E;77186.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858172 qtime=1364858239 etime=1364858172 start=1364858239 end=1364858263
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:24
02/04/2013 00:18:16;E;77187.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858182 qtime=1364858263 etime=1364858182 start=1364858263 end=1364858296
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:33
03/04/2013 07:23:02;E;77192.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907453 qtime=1364970153 etime=1364907453 start=1364970153 end=1364970182
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:29
03/04/2013 07:23:28;E;77193.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907469 qtime=1364970182 etime=1364907469 start=1364970182 end=1364970208
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
```

```
03/04/2013 07:23:51;E;77194.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907481 qtime=1364970208 etime=1364907481 start=1364970208 end=1364970231
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:23
04/04/2013 17:25:28;E;77747.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999418 qtime=1365092674 etime=1364999418 start=1365092674 end=1365092728
    Resource_List.nodes=1:ppn=4 resources_used.walltime=00:00:54
04/04/2013 23:10:27;E;77778.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1365000012 qtime=1365097808 etime=1365000012 start=1365097808 end=1365113427
    Resource_List.nodes=1:ppn=4 resources_used.walltime=04:20:19
05/04/2013 00:19:45;E;77776.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999966 qtime=1365097808 etime=1364999966 start=1365097808 end=1365117585
    Resource_List.nodes=1:ppn=4 resources_used.walltime=05:29:37
05/04/2013 00:50:14;E;77761.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999649 qtime=1365095656 etime=1364999649 start=1365095656 end=1365119414
    Resource_List.nodes=1:ppn=4 resources_used.walltime=06:35:58
05/04/2013 00:53:06;E;77775.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999921 qtime=1365097096 etime=1364999921 start=1365097096 end=1365119586
    Resource_List.nodes=1:ppn=4 resources_used.walltime=06:14:50
05/04/2013 02:03:15;E;79397.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365075452 qtime=1365123772 etime=1365075452 start=1365123772 end=1365123795
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:23
05/04/2013 02:05:22;E;79462.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365076875 qtime=1365123825 etime=1365076876 start=1365123825 end=1365123922
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:01:37
05/04/2013 02:08:31;E;79537.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365078333 qtime=1365123922 etime=1365078333 start=1365123922 end=1365124111
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:03:09
05/04/2013 07:44:28;E;77738.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999320 qtime=1365080728 etime=1364999320 start=1365080728 end=1365144268
    Resource_List.nodes=1:ppn=4 resources_used.walltime=17:39:00
05/04/2013 07:44:59;E;78715.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365027644 qtime=1365144268 etime=1365027644 start=1365144268 end=1365144299
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:31
05/04/2013 07:45:26;E;79221.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365071243 qtime=1365144299 etime=1365071243 start=1365144299 end=1365144326
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:27
05/04/2013 07:45:57;E;79251.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365072064 qtime=1365144326 etime=1365072064 start=1365144326 end=1365144357
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:31
05/04/2013 07:46:28;E;79348.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365074292 qtime=1365144357 etime=1365074292 start=1365144357 end=1365144388
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:31
```

```
05/04/2013 07:46:55;E;79365.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365074685 qtime=1365144388 etime=1365074685 start=1365144388 end=1365144415
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:27
05/04/2013 07:49:03;E;79409.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365075736 qtime=1365144415 etime=1365075736 start=1365144415 end=1365144543
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:08
05/04/2013 07:49:41;E;79432.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365076375 qtime=1365144543 etime=1365076375 start=1365144543 end=1365144581
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:38
05/04/2013 10:49:30;E;77733.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999241 qtime=1365079709 etime=1364999241 start=1365079709 end=1365155370
    Resource_List.nodes=1:ppn=4 resources_used.walltime=21:01:01
05/04/2013 14:05:26;E;77768.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999749 qtime=1365095741 etime=1364999749 start=1365095741 end=1365167126
    Resource_List.nodes=1:ppn=4 resources_used.walltime=19:49:45
01/05/2013 17:44:39;E;75246.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982454 qtime=1367426662 etime=1366982454 start=1367426662 end=1367426679
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:17
01/05/2013 17:45:10;E;75247.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982456 qtime=1367426696 etime=1366982456 start=1367426696 end=1367426710
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:14
01/05/2013 17:45:34;E;75250.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1367426710 etime=1366982457 start=1367426710 end=1367426734
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:24
01/05/2013 17:46:02;E;75249.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1367426734 etime=1366982457 start=1367426734 end=1367426762
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:28
01/05/2013 17:46:20;E;75248.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1367426762 etime=1366982457 start=1367426762 end=1367426780
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:18
01/05/2013 17:47:11;E;75251.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982458 qtime=1367426780 etime=1366982458 start=1367426780 end=1367426831
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:51
01/05/2013 17:47:46;E;75252.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982459 qtime=1367426831 etime=1366982459 start=1367426831 end=1367426866
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:35
02/05/2013 04:20:45;E;73269.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366898750 qtime=1367370314 etime=1366898750 start=1367370314 end=1367464845
    Resource_List.nodes=2:ppn=4 resources_used.walltime=26:15:31
02/05/2013 08:24:54;E;73002.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366889417 qtime=1367309583 etime=1366889417 start=1367309583 end=1367479494
    Resource_List.nodes=2:ppn=4 resources_used.walltime=47:11:51
```

```
03/05/2013 01:15:56;E;73207.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366897046 qtime=1367370226 etime=1366897046 start=1367370226 end=1367540156
    Resource_List.nodes=2:ppn=4 resources_used.walltime=47:12:10
06/05/2013 08:20:33;E;101864.eridani.qgg.hud.ac.uk;user=mduser2 queue=paraul ctime
    =1366706619 qtime=1367824815 etime=1366706619 start=1367824815 end=1367824833
    Resource_List.nodes=8:ppn=4 resources_used.walltime=00:00:18
06/05/2013 16:49:07;E;101863.eridani.qgg.hud.ac.uk;user=mduser2 queue=paraul ctime
    =1366640324 qtime=1367790539 etime=1366640324 start=1367790539 end=1367855347
    Resource_List.nodes=8:ppn=4 resources_used.walltime=18:00:08
```

F.2 Normalised data FCFS

```
02/04/2013 00:13:57;E;77181.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858011 qtime=1364858011 etime=1364858011 start=1364858011 end=1364858037
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
02/04/2013 00:14:23;E;77182.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858024 qtime=1364858037 etime=1364858024 start=1364858037 end=1364858063
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
02/04/2013 00:16:20;E;77183.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858154 qtime=1364858154 etime=1364858154 start=1364858154 end=1364858180
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
02/04/2013 00:16:46;E;77184.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858161 qtime=1364858180 etime=1364858161 start=1364858180 end=1364858206
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
02/04/2013 00:17:12;E;77185.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858167 qtime=1364858206 etime=1364858167 start=1364858206 end=1364858232
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
02/04/2013 00:17:38;E;77186.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858172 qtime=1364858232 etime=1364858172 start=1364858232 end=1364858258
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
02/04/2013 00:18:04;E;77187.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364858182 qtime=1364858258 etime=1364858182 start=1364858258 end=1364858284
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
03/04/2013 08:41:07;E;77192.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907453 qtime=1364974841 etime=1364907453 start=1364974841 end=1364974867
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
03/04/2013 08:41:15;E;77193.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907469 qtime=1364974849 etime=1364907469 start=1364974849 end=1364974875
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
```

```
03/04/2013 08:41:21;E;77194.eridani.qgg.hud.ac.uk;user=cfduser1 queue=paraul ctime
    =1364907481 qtime=1364974855 etime=1364907481 start=1364974855 end=1364974881
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:26
05/04/2013 10:47:20;E;77738.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999320 qtime=1365091700 etime=1364999320 start=1365091700 end=1365155240
    Resource_List.nodes=1:ppn=4 resources_used.walltime=17:39:00
05/04/2013 10:49:30;E;77733.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999241 qtime=1365079709 etime=1364999241 start=1365079709 end=1365155370
    Resource_List.nodes=1:ppn=4 resources_used.walltime=21:01:01
05/04/2013 16:01:19;E;77768.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999749 qtime=1365102694 etime=1364999749 start=1365102694 end=1365174079
    Resource_List.nodes=1:ppn=4 resources_used.walltime=19:49:45
05/04/2013 18:21:11;E;77747.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999418 qtime=1365099621 etime=1364999418 start=1365099621 end=1365182471
    Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50
05/04/2013 19:07:52;E;77761.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999649 qtime=1365102422 etime=1364999649 start=1365102422 end=1365185272
    Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50
05/04/2013 19:13:47;E;77776.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999966 qtime=1365102777 etime=1364999966 start=1365102777 end=1365185627
    Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50
05/04/2013 19:13:47;E;77775.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1364999921 qtime=1365102777 etime=1364999921 start=1365102777 end=1365185627
    Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50
05/04/2013 19:16:49;E;79397.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365075452 qtime=1365185653 etime=1365075452 start=1365185653 end=1365185809
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:02:36
05/04/2013 19:19:25;E;79462.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365076875 qtime=1365185809 etime=1365076876 start=1365185809 end=1365185965
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:02:36
05/04/2013 19:22:01;E;79537.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365078333 qtime=1365185965 etime=1365078333 start=1365185965 end=1365186121
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:02:36
06/04/2013 05:52:14;E;77778.eridani.qgg.hud.ac.uk;user=mduser2 queue=serialstd ctime
    =1365000012 qtime=1365141084 etime=1365000012 start=1365141084 end=1365223934
    Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50
07/04/2013 04:10:33;E;78715.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365027644 qtime=1365304077 etime=1365027644 start=1365304077 end=1365304233
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:36
07/04/2013 04:13:09;E;79221.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365071243 qtime=1365304233 etime=1365071243 start=1365304233 end=1365304389
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:36
```

```
07/04/2013 04:15:45;E;79251.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365072064 qtime=1365304389 etime=1365072064 start=1365304389 end=1365304545
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:36
07/04/2013 04:18:21;E;79348.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365074292 qtime=1365304545 etime=1365074292 start=1365304545 end=1365304701
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:36
07/04/2013 04:20:57;E;79365.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365074685 qtime=1365304701 etime=1365074685 start=1365304701 end=1365304857
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:36
07/04/2013 04:23:33;E;79409.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365075736 qtime=1365304857 etime=1365075736 start=1365304857 end=1365305013
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:36
07/04/2013 04:26:09;E;79432.eridani.qgg.hud.ac.uk;user=cfduser2 queue=paraul ctime
    =1365076375 qtime=1365305013 etime=1365076375 start=1365305013 end=1365305169
    Resource_List.nodes=3:ppn=4 resources_used.walltime=00:02:36
02/05/2013 17:48:43;E;75246.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982454 qtime=1367513280 etime=1366982454 start=1367513280 end=1367513323
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:43
02/05/2013 17:50:09;E;75247.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982456 qtime=1367513366 etime=1366982456 start=1367513366 end=1367513409
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:43
02/05/2013 17:50:52;E;75250.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1367513409 etime=1366982457 start=1367513409 end=1367513452
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:43
02/05/2013 17:51:35;E;75249.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1367513452 etime=1366982457 start=1367513452 end=1367513495
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:43
02/05/2013 17:52:18;E;75248.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982457 qtime=1367513495 etime=1366982457 start=1367513495 end=1367513538
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:43
02/05/2013 17:53:01;E;75251.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982458 qtime=1367513538 etime=1366982458 start=1367513538 end=1367513581
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:43
02/05/2013 17:53:44;E;75252.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366982459 qtime=1367513581 etime=1366982459 start=1367513581 end=1367513624
    Resource_List.nodes=2:ppn=4 resources_used.walltime=00:00:43
02/05/2013 21:02:56;E;73002.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366889417 qtime=1367396278 etime=1366889417 start=1367396278 end=1367524976
    Resource_List.nodes=2:ppn=4 resources_used.walltime=35:44:58
03/05/2013 06:59:51;E;73207.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366897046 qtime=1367432093 etime=1366897046 start=1367432093 end=1367560791
    Resource_List.nodes=2:ppn=4 resources_used.walltime=35:44:58
```

```
03/05/2013 07:04:02;E;73269.eridani.qgg.hud.ac.uk;user=mduser1 queue=parastd ctime
    =1366898750 qtime=1367432344 etime=1366898750 start=1367432344 end=1367561042
    Resource_List.nodes=2:ppn=4 resources_used.walltime=35:44:58
06/05/2013 08:53:48;E;101864.eridani.qgg.hud.ac.uk;user=mduser2 queue=paraul ctime
    =1366706619 qtime=1367826811 etime=1366706619 start=1367826811 end=1367826828
    Resource_List.nodes=8:ppn=4 resources_used.walltime=00:00:17
07/05/2013 20:33:13;E;101863.eridani.qgg.hud.ac.uk;user=mduser2 queue=paraul ctime
    =1366640324 qtime=1367793937 etime=1366640324 start=1367793937 end=1367955193
    Resource_List.nodes=8:ppn=4 resources_used.walltime=44:47:36
```

F.3 Normalised data Moulded

```
02/04/2013 00:37:43;E;77181.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case237 queue=
    paraul ctime=1364858011 qtime=1364859462 etime=1364858011 start=1364859462 end
    =1364859463 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:44;E;77182.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case1 queue=
    paraul ctime=1364858024 qtime=1364859463 etime=1364858024 start=1364859463 end
    =1364859464 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:45;E;77183.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case238 queue=
    paraul ctime=1364858154 qtime=1364859464 etime=1364858154 start=1364859464 end
    =1364859465 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:46;E;77184.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case239 queue=
    paraul ctime=1364858161 qtime=1364859465 etime=1364858161 start=1364859465 end
    =1364859466 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:47;E;77185.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case240 queue=
    paraul ctime=1364858167 qtime=1364859466 etime=1364858167 start=1364859466 end
    =1364859467 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 00:37:48;E;77186.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case241 queue=
    paraul ctime=1364858172 qtime=1364859467 etime=1364858172 start=1364859467 end
    =1364859468 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
```

```
02/04/2013 00:37:49;E;77187.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case242 queue=
    paraul ctime=1364858182 qtime=1364859468 etime=1364858182 start=1364859468 end
    =1364859469 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 13:57:34;E;77192.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case239 queue=
    paraul ctime=1364907453 qtime=1364907453 etime=1364907453 start=1364907453 end
    =1364907454 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 13:57:50;E;77193.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case240 queue=
    paraul ctime=1364907469 qtime=1364907469 etime=1364907469 start=1364907469 end
    =1364907470 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
02/04/2013 13:58:02;E;77194.eridani.qgg.hud.ac.uk;user=cfduser1 jobname=case241 queue=
    paraul ctime=1364907481 qtime=1364907481 etime=1364907481 start=1364907481 end
    =1364907482 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:00:01 app_name=
    fluent iterations=1 dataset_size=16000000
04/04/2013 14:28:11;E;77733.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999241 qtime=1364999241 etime=1364999241 start=1364999241 end
    =1365082091 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:29:30;E;77738.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999320 qtime=1364999320 etime=1364999320 start=1364999320 end
    =1365082170 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:30:30;E;78715.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365027644 qtime=1365082132 etime=1365027644 start=1365082132
    end=1365082230 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:31:08;E;77747.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999418 qtime=1364999418 etime=1364999418 start=1364999418 end
    =1365082268 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:32:09;E;79221.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365071243 qtime=1365082231 etime=1365071243 start=1365082231
    end=1365082329 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:32:46;E;79251.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365072064 qtime=1365082268 etime=1365072064 start=1365082268
    end=1365082366 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
```

```
04/04/2013 14:33:23;E;79348.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365074292 qtime=1365082305 etime=1365074292 start=1365082305
    end=1365082403 Resource_List.nodes=1:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:33:47;E;79365.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365074685 qtime=1365082329 etime=1365074685 start=1365082329
    end=1365082427 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:34:24;E;79397.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365075452 qtime=1365082366 etime=1365075452 start=1365082366
    end=1365082464 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:34:59;E;77761.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999649 qtime=1364999649 etime=1364999649 start=1364999649 end
    =1365082499 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:35:21;E;79409.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365075736 qtime=1365082423 etime=1365075736 start=1365082423
    end=1365082521 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:36:39;E;77768.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999749 qtime=1364999749 etime=1364999749 start=1364999749 end
    =1365082599 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:37:12;E;79432.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365076375 qtime=1365082534 etime=1365076375 start=1365082534
    end=1365082632 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:39:31;E;77775.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999921 qtime=1364999921 etime=1364999921 start=1364999921 end
    =1365082771 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:40:16;E;77776.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1364999966 qtime=1364999966 etime=1364999966 start=1364999966 end
    =1365082816 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
04/04/2013 14:41:02;E;77778.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    serialstd ctime=1365000012 qtime=1365000012 etime=1365000012 start=1365000012 end
    =1365082862 Resource_List.nodes=1:ppn=4 resources_used.walltime=23:00:50 app_name=
    dlpoly iterations=5000000 dataset_size=500
```

```
04/04/2013 14:41:09;E;79462.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365076875 qtime=1365082771 etime=1365076876 start=1365082771
    end=1365082869 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
04/04/2013 14:42:47;E;79537.eridani.qgg.hud.ac.uk;user=cfduser2 jobname=60000-0.262
    queue=paraul ctime=1365078333 qtime=1365082869 etime=1365078333 start=1365082869
    end=1365082967 Resource_List.nodes=3:ppn=4 resources_used.walltime=00:01:38
    app_name=fluent iterations=12 dataset_size=8100000
24/04/2013 12:24:51;E;101864.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    paraul ctime=1366706619 qtime=1366802674 etime=1366706619 start=1366802674 end
    =1366802691 Resource_List.nodes=1:ppn=4 resources_used.walltime=00:00:17 app_name=
    dlpoly iterations=100 dataset_size=8000
26/04/2013 09:12:10;E;101863.eridani.qgg.hud.ac.uk;user=mduser2 jobname=jcl queue=
    paraul ctime=1366640324 qtime=1366802674 etime=1366640324 start=1366802674 end
    =1366963930 Resource_List.nodes=4:ppn=4 resources_used.walltime=44:47:36 app_name=
    dlpoly iterations=1000000 dataset_size=8000
27/04/2013 06:39:18;E;73002.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366889417 qtime=1366968592 etime=1366889417 start=1366968592 end
    =1367041158 Resource_List.nodes=4:ppn=4 resources_used.walltime=20:09:26 app_name=
    dlpoly iterations=900000 dataset_size=4000
27/04/2013 08:01:01;E;75246.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982454 qtime=1367046036 etime=1366982454 start=1367046036 end
    =1367046061 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:01:26;E;75247.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982456 qtime=1367046061 etime=1366982456 start=1367046061 end
    =1367046086 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:01:37;E;75250.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982457 qtime=1367046072 etime=1366982457 start=1367046072 end
    =1367046097 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:01:51;E;75249.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982457 qtime=1367046086 etime=1366982457 start=1367046086 end
    =1367046111 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:02:02;E;75248.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982457 qtime=1367046097 etime=1366982457 start=1367046097 end
    =1367046122 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
```

```
27/04/2013 08:02:16;E;75251.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982458 qtime=1367046111 etime=1366982458 start=1367046111 end
    =1367046136 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 08:02:27;E;75252.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366982459 qtime=1367046122 etime=1366982459 start=1367046122 end
    =1367046147 Resource_List.nodes=4:ppn=4 resources_used.walltime=00:00:25 app_name=
    dlpoly iterations=300 dataset_size=4000
27/04/2013 21:06:03;E;73207.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366897046 qtime=1367020597 etime=1366897046 start=1367020597 end
    =1367093163 Resource_List.nodes=4:ppn=4 resources_used.walltime=20:09:26 app_name=
    dlpoly iterations=900000 dataset_size=4000
27/04/2013 21:18:13;E;73269.eridani.qgg.hud.ac.uk;user=mduser1 jobname=dlpoly queue=
    parastd ctime=1366898750 qtime=1367021327 etime=1366898750 start=1367021327 end
    =1367093893 Resource_List.nodes=4:ppn=4 resources_used.walltime=20:09:26 app_name=
    dlpoly iterations=900000 dataset_size=4000
```

References

Agarwala, S., Poellabauer, C., Kong, J., Schwan, K., & Wolf, M. (2003, Sep). System-level resource monitoring in high-performance computing environments. *Journal of Grid Computing*, *1*(3), 273 – 289. doi: 10.1023/B:GRID.0000035189.80518.5d

Alam, S. R., Barrett, R. F., Kuehn, J. A., Roth, P. C., & Vetter, J. S. (2006). Characterization of scientific workloads on systems with multi-core processors. In *Workload characterization, 2006 IEEE international symposium on* (pp. 225–236). IEEE. Retrieved 2014-12-31, from `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4086151`

AlJahdali, H., Albatli, A., Garraghan, P., Townend, P., Lau, L., & Xu, J. (2014). Multi-tenancy in cloud computing. In (pp. 344–351). IEEE. Retrieved 2015-01-31, from `http://eprints.whiterose.ac.uk/80819/` doi: 10.1109/SOSE.2014.50

Aljahdali, H., Townend, P., & Xu, J. (2013). Enhancing multi-tenancy security in the cloud IaaS model over public deployment. In *2013 IEEE 7th international symposium on service oriented system engineering (SOSE)* (pp. 385–390). doi: 10.1109/SOSE.2013.50

Allen, B., Bresnahan, J., Childers, L., Foster, I., Kandaswamy, G., Kettimuthu, R., ... others (2012). Software as a service for data scientists. *Communications of the ACM*, *55*(2), 81–88.

Alvarruiz, F., de Alfonso, C., Caballer, M., & Hern'ndez, V. (2012). An energy manager for high performance computer clusters. In *Parallel and distributed processing with applications (ispa), 2012 ieee 10th international symposium on* (pp. 231–238).

Amit, G., Caspi, Y., Vitale, R., & Pinhas, A. T. (2006). Scalability of multimedia applications on next-generation processors. In *Multimedia and expo, 2006 IEEE international conference on* (pp. 17–20). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4036525

Anderson, D. P. (2004). Boinc: A system for public-resource computing and storage. In *Grid computing, 2004. proceedings. fifth ieee/acm international workshop on* (pp. 4–10).

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... others (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58. Retrieved 2014-12-31, from http://m.cacm.acm.org/magazines/2010/4/81493-a-view-of-cloud-computing/fulltext

A Vouk, M. (2008). Cloud computing–issues, research and implementations. *CIT. Journal of Computing and Information Technology*, *16*(4), 235–246. Retrieved 2014-12-31, from http://hrcak.srce.hr/file/69202

Bayucan, A., & Henderson, R. L. (2000). *Portable batch system administrator guide release: OpenPBS 2.3. 16*.

Bernstein, J., & McMahon, K. (2012). Computing on demand—hpc as a service. *Penguin Computing*.

Bientinesi, P., Iakymchuk, R., & Napper, J. (2010). Hpc on competitive cloud resources. In *Handbook of cloud computing* (pp. 493–516). Springer.

Boneti, C., Gioiosa, R., Cazorla, F. J., & Valero, M. (2008). A dynamic scheduler for balancing HPC applications. In *Proceedings of the 2008 ACM/IEEE conference on supercomputing* (p. 41). IEEE Press. Retrieved 2014-12-31, from http://dl.acm.org/citation.cfm?id=1413412

Bonner, S., Antoniou, G., Kureshi, I., Moss, L., Corsar, D., & TachmazidisLysik, I. (2014). Using hadoop to implement a semantic method of assessing the quality of research medical datasets. In *Big data science 2014, the third ase international conference on.* IEEE.

Bonner, S., Pulley, C., Kureshi, I., Holmes, V., Brennan, J., & James, Y. (2013). Using OpenStack to improve student experience in an HE environment. In *Science and information conference (SAI), 2013* (pp. 888–893). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6661847

Brandt, J., Gentile, A., Mayo, J., Pebay, P., Roe, D., Thompson, D., & Wong, M. (2009). Resource monitoring and management with OVIS to enable HPC in cloud computing environments. In *Parallel & distributed processing, 2009. IPDPS 2009. IEEE international symposium on* (pp. 1–8). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5161234

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. Retrieved 2014-12-31, from http://eprints.hud.ac.uk/17337

Brennan, J., Kureshi, I., & Holmes, V. (2014). Cdes: An approach to hpc workload modelling. In *Proceedings of the 2014 ieee/acm 18th international symposium on distributed simulation and real time applications* (pp. 47–54).

Brennan, J. D. (2014). *Developing a trusted computational grid* (Master's thesis, University of Huddersfield). Retrieved from http://eprints.hud.ac.uk/23308/

Brodkin, J. (2011). $1,279-per-hour, 30,000 core cluster built on amazon EC2 cloud. *Ars Technica*, *20*.

Buyya, R., & Murshed, M. (2002). 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 2014-12-31, from http://onlinelibrary.wiley.com/doi/10.1002/cpe.710/abstract

Buyya, R., Yeo, C. S., & Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High performance computing and communications, 2008. HPCC'08. 10th IEEE international conference on* (pp. 5–13). Ieee. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4637675

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, *25*(6), 599–616. Retrieved 2014-12-31, from http://www.sciencedirect.com/science/article/pii/S0167739X08001957

Carroll, T. E., & Grosu, D. (2010). Incentive compatible online scheduling of malleable parallel jobs with individual deadlines. In *Parallel processing (ICPP), 2010 39th international conference on* (pp. 516–524). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5599194

Chen, X., Lu, C.-D., & Pattabiraman, K. (2014). Failure analysis of jobs in compute clouds: A google cluster case study. In *the international symposium on software reliability engineering (issre). ieee.*

Chieu, T. C., Mohindra, A., Karve, A. A., & Segal, A. (2009). Dynamic scaling of web applications in a virtualized cloud computing environment. In *e-business engineering, 2009. icebe'09. ieee international conference on* (pp. 281–286).

Cirne, W., & Berman, F. (2001). A model for moldable supercomputer jobs.

In *Parallel and distributed processing symposium., proceedings 15th international* (pp. 8–pp). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=925004

Cirne, W., & Berman, F. (2002). Using moldability to improve the performance of supercomputer jobs. *Journal of Parallel and Distributed Computing*, *62*(10), 1571–1601. Retrieved 2014-12-31, from http://www.sciencedirect.com/science/article/pii/S0743731502918691

Computing, A., & Computing, G. (2012). Torque resource manager. *online] http://www. adaptivecomputing. com*.

Cybenko, G., Kipp, L., Pointer, L., & Kuck, D. (1990). *Supercomputer performance evaluation and the perfect benchmarks* (Vol. 18) (No. 3b). ACM. Retrieved 2014-12-31, from http://dl.acm.org/citation.cfm?id=255163

da Silva, F., & Senger, H. (2010). Scalability analysis of embarassingly parallel applications on large clusters. In *Parallel & distributed processing, workshops and phd forum (IPDPSW), 2010 IEEE international symposium on* (pp. 1–8). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5470724

De Alfonso, C., Caballer, M., & Hernández, V. (2010). Efficient power management in high performance computer clusters. In *1st international multiconference on innovative developments in ict* (pp. 39–44).

de Gyves Avila, S., & Djemame, K. (2013). Fuzzy logic based QoS optimization mechanism for service composition. In (pp. 182–191). IEEE. Retrieved 2015-01-31, from http://eprints.whiterose.ac.uk/79907/ doi: 10.1109/SOSE.2013.28

Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., & Vakali, A. (2009). Cloud computing: distributed internet computing for IT and scientific research. *Internet Computing, IEEE*, *13*(5), 10–13. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5233607

Dongarra, J. (2013). *November 2013 | the green500.* Retrieved 2014-12-31, from http://www.green500.org/lists/green201311

Dongarra, J. (2014). *June 2014 | TOP500 supercomputer sites.* Retrieved 2014-12-31, from http://www.top500.org/list/2014/11/

Dongarra, J. J., Bunch, J. R., Moler, C. B., & Stewart, G. W. (1979). *Linpack users' guide* (Vol. 8). Siam.

Downey, A. B. (1997). *A model for speedup of parallel programs*. University of California, Berkeley, Computer Science Division. Retrieved 2014-12-31, from http://www.eecs.berkeley.edu/Pubs/TechRpts/1997/CSD-97-933.pdf

Eigenmann, R., & Hassanzadeh, S. (1996). Benchmarking with real industrial applications: the SPEC high-performance group. *Computing in Science and Engineering*, *3*(1), 18–23. Retrieved 2014-12-31, from http://www.computer.org/csdl/mags/cs/1996/01/c1018.pdf

Elton, B. H., Samsi, S., Smith, H. B., Humphrey, L., Guilfoos, B., Ahalt, S., ... others (2009). A scalability study (as a guide for HPC operations at a remote test facility) on DSRC HPC systems of radio frequency tomography code written for MATLAB® and parallelized via star-p®. In *DoD high performance computing modernization program users group conference (HPCMP-UGC), 2009* (pp. 401–409). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5729498

Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., & Wong, P. (1997). Theory and practice in parallel job scheduling. In *Job scheduling strategies for parallel processing* (pp. 1–34). Springer. Retrieved 2014-12-31, from http://link.springer.com/chapter/10.1007/3-540-63574-2_14

Foster, I. (2011). Accelerating and democratizing science through cloud-based services. *IEEE Internet Comput.*, *15*(ANL/MCS/JA-69753).

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud computing and grid

computing 360-degree compared. In *Grid computing environments workshop, 2008. GCE'08* (pp. 1–10). Ieee. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4738445

Goel, B., McKee, S. A., Gioiosa, R., Singh, K., Bhadauria, M., & Cesati, M. (2010). Portable, scalable, per-core power estimation for intelligent resource management. In *Green computing conference, 2010 international* (pp. 135–146). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5598313

Grossman, R. L. (2009). The case for cloud computing. *IT professional*, *11*(2), 23–27. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4804045

Gubb, D., Holmes, V., Kureshi, I., Liang, S., & James, Y. (2012, September). Implementing a condor pool using a green-it policy.. Retrieved from http://eprints.hud.ac.uk/15839/

Hamscher, V., Schwiegelshohn, U., Streit, A., & Yahyapour, R. (2000). Evaluation of job-scheduling strategies for grid computing. In *Grid computing—GRID 2000* (pp. 191–202). Springer. Retrieved 2014-12-31, from http://link.springer.com/chapter/10.1007/3-540-44444-0_18

Hayes, B. (2008, July). Cloud computing. *Commun. ACM*, *51*(7), 9–11. Retrieved 2014-12-31, from http://doi.acm.org/10.1145/1364782.1364786 doi: 10.1145/1364782.1364786

He, Q., Zhou, S., Kobler, B., Duffy, D., & McGlynn, T. (2010). Case study for running hpc applications in public clouds. In *Proceedings of the 19th acm international symposium on high performance distributed computing* (pp. 395–401).

Holmes, V., & Kureshi, I. (2010). Huddersfield university campus grid: Qgg of oscar clusters. In *Journal of physics: conference series* (Vol. 256, p. 012022). IOP Publishing. Retrieved 2014-12-31, from http://iopscience.iop.org/1742-6596/256/1/012022

Huang, K.-C., Shih, P.-C., & Chung, Y.-C. (2009). Adaptive processor allocation for moldable jobs in computational grid. *International Journal of Grid and High Performance Computing (IJGHPC)*, *1*(1), 10–21. Retrieved 2014-12-31, from http://www.igi-global.com/article/international-journal-grid-high-performance/2165

Hungershofer, J. (2004). On the combined scheduling of malleable and rigid jobs. In *Computer architecture and high performance computing, 2004. SBAC-PAD 2004. 16th symposium on* (pp. 206–213). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1364755

Iqbal, S., Gupta, R., & Fang, Y.-C. (2005). Planning considerations for job scheduling in HPC clusters. *Hgh-Performance Computing, reprinted from Dell Power Solutions*.

Jackson, D., Snell, Q., & Clement, M. (2001). Core algorithms of the maui scheduler. In *Job scheduling strategies for parallel processing* (pp. 87–102).

Jie, Y., Qiu, J., & Li, Y. (2009). A profile-based approach to just-in-time scalability for cloud applications. In *Cloud computing, 2009. CLOUD'09. IEEE international conference on* (pp. 9–16). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5284077

Kalé, L. V., Kumar, S., & DeSouza, J. (2002). A malleable-job system for timeshared parallel machines. In *Cluster computing and the grid, 2002. 2nd IEEE/ACM international symposium on* (pp. 230–230). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1540460

Kelly, A. (2014). *UNIVERSITIES RISE TO THE EFFICIENCY CHALLENGE.* Retrieved 2014-12-31, from http://www.ncl.ac.uk/press.office/press.release/item/universities-rise-to-the-efficiencychallenge

Kenway, C. P. M. S. P. O. . P. M., R. (2012). *UK participation in high performance computing (HPC) at the european level - publications - GOV.UK.* Retrieved 2014-12-31, from https://www.gov.uk/government/publications/uk-participation-in-high-performance-computing-hpc-at-the-european-level

Kerbyson, D. J., Alme, H. J., Hoisie, A., Petrini, F., Wasserman, H. J., & Gittings, M. (2001). Predictive performance and scalability modeling of a large-scale application. In *Proceedings of the 2001 ACM/IEEE conference on supercomputing (CDROM)* (pp. 37–37). ACM. Retrieved 2014-12-31, from http://dl.acm.org/citation.cfm?id=582071

Khalid, O., Maljevic, I., Anthony, R., Petridis, M., Parrott, K., & Schulz, M. (2009). Dynamic scheduling of virtual machines running hpc workloads in scientific grids. In *New technologies, mobility and security (NTMS), 2009 3rd international conference on* (pp. 1–5). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5384725

Krallmann, J., Schwiegelshohn, U., & Yahyapour, R. (1999). On the design and evaluation of job scheduling algorithms. In *Job scheduling strategies for parallel processing* (pp. 17–42).

Kureshi, I. (2010). *Establishing a university grid for HPC applications* (Doctoral dissertation, University of Huddersfield). Retrieved 2014-12-31, from http://eprints.hud.ac.uk/10169

Kureshi, I., Holmes, V., & Cooke, D. J. (2012). Robust mouldable scheduling using application benchmarking for elastic enviornments. In *Local proceedings of the fifth balkan conference in informatics* (pp. 51–57). University of Novi Sad, Serbia. Retrieved 2014-12-31, from http://eprints.hud.ac.uk/15025

Kureshi, I., Pulley, C., Brennan, J., Holmes, V., Bonner, S., & James, Y. (2013). Advancing research infrastructure using OpenStack. *International Journal of Advanced Computer Science and Applications*, *3*(4), 64–70. Retrieved

2014-12-31, from http://eprints.hud.ac.uk/19421

Laure, E., Edlund, A., Pacini, F., Buncic, P., Barroso, M., Di Meglio, A., ... others (2006). *Programming the grid with glite* (Tech. Rep.).

Legrand, A., Marchal, L., & Casanova, H. (2003). Scheduling distributed applications: the simgrid simulation framework. In *Cluster computing and the grid, 2003. proceedings. CCGrid 2003. 3rd IEEE/ACM international symposium on* (pp. 138–145). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1199362

Mao, M., Li, J., & Humphrey, M. (2010). Cloud auto-scaling with deadline and budget constraints. In *Grid computing (grid), 2010 11th ieee/acm international conference on* (pp. 41–48).

Marshall, P., Keahey, K., & Freeman, T. (2010). Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing* (pp. 43–52). IEEE Computer Society. Retrieved 2014-12-31, from http://dl.acm.org/citation.cfm?id=1845214

Meredith, M., Carrigan, T., Brockman, J., Cloninger, T., Privoznik, J., & Williams, J. (2003). Exploring beowulf clusters. *Journal of Computing Sciences in Colleges*, *18*(4), 268–284. Retrieved 2014-12-31, from http://dl.acm.org/citation.cfm?id=767641

Mönch, L., Balasubramanian, H., Fowler, J. W., & Pfund, M. E. (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research*, *32*(11), 2731–2750.

Montero, R. S., Huedo, E., & Llorente, I. M. (2006). Benchmarking of high throughput computing applications on grids. *Parallel Computing*, *32*(4), 267–279. Retrieved 2014-12-31, from http://www.sciencedirect.com/science/article/pii/S0167819105001651

Naik, V. K. (1992). Scalability issues for a class of CFD applications. In *Scalable*

high performance computing conference, 1992. SHPCC-92, proceedings. (pp. 268–275). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=232632

Newall, M., Holmes, V., & Lunn, P. (2014). GPU cluster for accelerating processing and visualisation of scientific and engineering data. In *Proceedings of the science and information conference.* IEEE. Retrieved 2014-12-31, from http://eprints.hud.ac.uk/21907/

Novotny, J., Tuecke, S., & Welch, V. (2001). An online credential repository for the grid: Myproxy. In *High performance distributed computing, 2001. proceedings. 10th ieee international symposium on* (pp. 104–111).

Nurmi, D., Mandal, A., Brevik, J., Koelbel, C., Wolski, R., & Kennedy, K. (2006). Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *Proceedings of the 2006 ACM/IEEE conference on supercomputing* (p. 119). ACM. Retrieved 2014-12-31, from http://dl.acm.org/citation.cfm?id=1188579

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., & Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In *Cluster computing and the grid, 2009. ccgrid'09. 9th ieee/acm international symposium on* (pp. 124–131).

Oprescu, A., & Kielmann, T. (2010). Bag-of-tasks scheduling under budget constraints. In *Cloud computing technology and science (CloudCom), 2010 IEEE second international conference on* (pp. 351–359). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5708470

Padgett, J., Djemame, K., & Dew, P. (2005). Grid-based SLA management. In *Advances in grid computing-EGC 2005* (pp. 1076–1085). Springer. Retrieved 2014-12-31, from http://link.springer.com/chapter/10.1007/11508380_110

Pandey, S., Wu, L., Guru, S. M., & Buyya, R. (2010). A particle swarm

optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced information networking and applications (AINA), 2010 24th IEEE international conference on* (pp. 400–407). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5474725

Parashar, M., & Hariri, S. (1997). Interpretive performance prediction for high performance application development. In *System sciences, 1997, proceedings of the thirtieth hawaii international conference on* (Vol. 1, pp. 462–471). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=667300

Petitet, A. (2004). HPL-a portable implementation of the high-performance linpack benchmark for distributed-memory computers. *http://www. netlib-. org/-benchmark/hpl/* . Retrieved 2014-12-31, from http://ci.nii.ac.jp/naid/10011009158/

Ramakrishnan, L., Koelbel, C., Kee, Y.-S., Wolski, R., Nurmi, D., Gannon, D., ... others (2009). VGrADS: enabling e-science workflows on grids and clouds with fault tolerance. In *High performance computing networking, storage and analysis, proceedings of the conference on* (pp. 1–12). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6375523

Rasheed, H., Gruber, R., Keller, V., Waldrich, O., Ziegler, W., Wieder, P., ... Kunszt, P. (2007). Ianos: An intelligent application oriented scheduling middleware for a hpc grid. *Institute on Resource Management and Scheduling, CoreGRID-Network of Excellence, Tech. Rep. TR-0110*. Retrieved 2014-12-31, from http://www.researchgate.net/publication/228368181_Ianos_An_intelligent_application_oriented_scheduling_middleware_for_a_hpc_grid/file/9fcfd50ed31eaab7ee.pdf

Said, M. F. M., Taib, M. N., & Yahya, S. (2008). Analysis of the CPU utilization for point-to-point communication operations in a beowulf cluster system.

In *Information technology, 2008. ITSim 2008. international symposium on* (Vol. 1, pp. 1–6). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4631592

Saule, E., Bozdağ, D., & Catalyurek, U. V. (2010). A moldable online scheduling algorithm and its application to parallel short sequence mapping. In *Job scheduling strategies for parallel processing* (pp. 93–109). Springer. Retrieved 2014-12-31, from http://link.springer.com/chapter/10.1007/978-3-642-16505-4_6

Sayeed, M., Bae, H., Zheng, Y., Armstrong, B., Eigenmann, R., & Saied, G. (2008). Measuring high-performance computing with real applications. *Computing in Science & Engineering*, *10*(4), 60–70. Retrieved 2014-12-31, from http://scitation.aip.org/content/aip/journal/cise/10/4/10.1109/MCSE.2008.98

Schwan, K., Cooper, B. F., Eisenhauer, G. S., Gavrilovska, A., Wolf, M., Abbasi, H., ... others (2005). Autonomic information flows.

Simon, T. A., Cable, S. B., & Mahmoodi, M. (2007). Application scalability and performance on multicore architectures. In *DoD high performance computing modernization program users group conference, 2007* (pp. 378–381). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4438014

Sloan, J. D. (2004). *High performance linux clusters with oscar, rocks, openmosix, and mpi*. " O'Reilly Media, Inc.".

Smith, W., & Todorov, I. T. (2006). A short description of dl_poly. *Molecular Simulation*, *32*(12-13), 935–943.

Snavely, A., Carrington, L., Wolter, N., Labarta, J., Badia, R., & Purkayastha, A. (2002). A framework for performance modeling and prediction. In *Supercomputing, ACM/IEEE 2002 conference* (pp. 21–21). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1592857

Srinivasan, S., Krishnamoorthy, S., & Sadayappan, P. (2003). A robust scheduling technology for moldable scheduling of parallel jobs. In *Cluster computing, 2003. proceedings. 2003 IEEE international conference on* (pp. 92–99). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1253304

Sterling, T. L. (2002). *Beowulf cluster computing with linux*. MIT press. Retrieved 2014-12-31, from http://books.google.co.uk/books?hl=en&lr=&id=M73h9u9Q7sAC&oi=fnd&pg=PA1&dq=Beowulf+Cluster+Computing+with+Linux&ots=utTBFKwyVQ&sig=1yoSPWQ1HGXZOVkWUpxG3dAXoro

Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, S. K., Gehrke, J. E., & Plaxton, C. G. (1996). A proportional share resource allocation algorithm for real-time, time-shared systems. In *Real-time systems symposium, 1996., 17th IEEE* (pp. 288–299). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=563725

Stumm, M. (1988). The design and implementation of a decentralized scheduling facility for a workstation cluster. In *Computer workstations, 1988., proceedings of the 2nd IEEE conference on* (pp. 12–22). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4797

Tanenbaum, A. S., & Tannenbaum, A. (1992). *Modern operating systems* (Vol. 2). Prentice hall Englewood Cliffs.

Trader, T. (2012). *Cycle spins up 50,000-core cluster in amazon cloud* (Tech. Rep.).

Trystram, D. (2001). Scheduling parallel applications using malleable tasks on clusters. In *Parallel and distributed processing symposium, international* (Vol. 3, pp. 30199a–30199a). IEEE Computer Society. Retrieved 2014-12-31, from http://www.computer.org/csdl/proceedings/ipdps/2001/0990/03/099030199a.pdf

Van den Bossche, R., Vanmechelen, K., & Broeckhove, J. (2010). Cost-optimal

scheduling in hybrid iaas clouds for deadline constrained workloads. In *Cloud computing (cloud), 2010 ieee 3rd international conference on* (pp. 228–235).

Vaquero, L. M., Rodero-Merino, L., & Buyya, R. (2011). Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, *41*(1), 45–52.

Vaquero, L. M., Rodero-Merino, L., Caceres, J., & Lindner, M. (2008). A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, *39*(1), 50–55. Retrieved 2014-12-31, from http://dl.acm.org/citation.cfm?id=1496100

Wallom, D. (2010). *The NGS cloud pilots, their structure, users and lessons learned.* Retrieved from http://web3-test.esc.rl.ac.uk/sites/default/files/file/ngsif10%20presentations/Clouds%20IF10.pdf.

Wang, L., Tao, J., Kunze, M., Castellanos, A. C., Kramer, D., & Karl, W. (2008). Scientific cloud computing: Early definition and experience. In *HPCC* (Vol. 8, pp. 825–830). Retrieved 2014-12-31, from http://cyberaide.googlecode.com/svn/trunk/papers/08-cloud/vonLaszewski-08-cloud.pdf

Wang, L.-X. (1999). *A course in fuzzy systems*. Prentice-Hall press, USA.

Weiss, A. (2007). Computing in the clouds. *networker*, *11*(4). Retrieved 2014-12-31, from http://di.ufpe.br/~redis/intranet/bibliography/middleware/weiss-computing08.pdf

Weng, C., & Lu, X. (2005). Heuristic scheduling for bag-of-tasks applications in combination with qos in the computational grid. *Future Generation Computer Systems*, *21*(2), 271–280.

Wilson, S. (2010). *Grid engine: The world's first cloud-aware distributed resource manager (virtual steve).* Retrieved 2014-12-31, from https://blogs.oracle.com/stevewilson/entry/grid_engine_the_world_s

Woo, S.-H., Yang, S.-B., Kim, S.-D., & Han, T.-D. (1997). Task scheduling in distributed computing systems with a genetic algorithm. In *High performance computing on the information superhighway, 1997. HPC asia'97* (pp. 301–305). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=592164

Yao, F., Demers, A., & Shenker, S. (1995). A scheduling model for reduced CPU energy. In *Foundations of computer science, 1995. proceedings., 36th annual symposium on* (pp. 374–382). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=492493

Zadeh, L. A. (1994). The role of fuzzy logic in modeling, identification and control. *Modeling, identification and control*, *15*(3), 191–203.

Zaman, S., & Grosu, D. (2011). Efficient bidding for virtual machine instances in clouds. In *Cloud computing (CLOUD), 2011 IEEE international conference on* (pp. 41–48). IEEE. Retrieved 2014-12-31, from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6008691

Glossary

Batch Processing System / **Batch System** / **Offline Processing**: This category of systems take all inputs for several jobs and once its buffer is full or an operator instructs it to then the execution of commands begins. Jobs submitted will never run at submit time.

Batch Queuing System: The evolution of the a Batch Processing System bridging the gap to Online systems. If an online system does not have the resources to begin full execution it uses a batch queueing system to streamline pending jobs - the system is still defined as online because the job is considered for execution but the results are delayed.

Fair share: A system of scheduling that prevents one user or group from consuming all resources. If the resources are uncontested then a single user or group gets full access but once others start to request resources then the first user or groups resources are taken away to create an equitable distribution.

Malleable Job: A job whose allocated resources can be changed while it is running. e.g. during read-in or write-out (inherently serial operations) a single node can be allocated, and during processing or MPI stages more resources can be assigned.

Mouldable Job: A job whole allocated resources can be changed and decided before execution begins. e.g. allocating 2 CPUs rather that the 4 requested to enable faster turn-around.

Online System/Processing: An online system is one that reacts immediate to a stimulus however the time bound is not strictly enforced. e.g. a job when submitted runs immediately but results and responses are not immediate.

Quality of Service (QoS): Quality of Service is a measure of how quickly a user submitted job can be turned around. QoS is assessed based on the increase/decrease of the time a job needs to wait for required resources before it can begin. QoS and wait time are inversely related where an increased wait time implies a reduction in quality of service.

Real Time System: A real-time system is one that must process information and output results within a specified time and is usually considered responsive or immediate. If the time limits are exceeded it is considered a failure. e.g. an instruction given to a program must happen immediately else the system is ”stuck” (clicking the Start button in Windows).

Run Time: Used interchangeably as the time at which a queued job begins to run (e.g. at run time) or the duration of the jobs execution cycle (not including the wait time)(e.g. the job has a 3 hour run time).

Submit Time: The time at which a user submits their job to the HPC cluster.

Turn-around-Time: The total time from Submit time till the job is evicted from the system on completion.

Wait Time: The length of time a job waits in the queue before execution can begin. This is typically due to resources not being available but can also be caused by dependancies to early executing jobs.

Wall Time: A term from the Terascale Open-Source Resource and QUEue Manager (TORQUE) and Portable Batch System (PBS) job management system. It refers to the length of time the job has been running i.e. duration of run time. Similar to Run Time