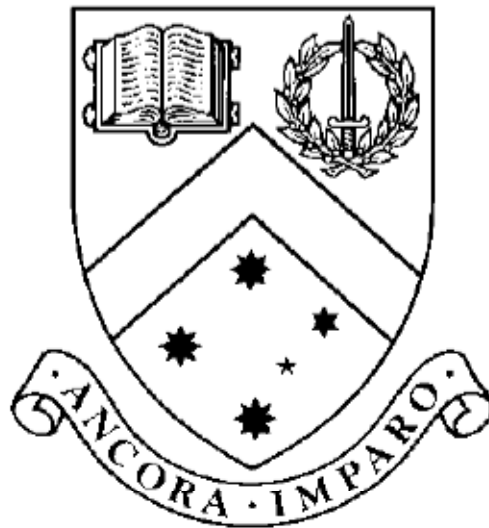


Dynamic Generation and Execution of Virtual Machine Work Units on the Grid

by

Timothy H. Ward



Thesis

Submitted by Timothy H. Ward

in partial fulfillment of the Requirements for the Degree of
Bachelor of Software Engineering with Honours (2770)

Supervisor: David Abramson and Wojtek Goscinski

**Clayton School of Information Technology
Monash University**

October, 2008

© Copyright
by
Timothy H. Ward
2008

Dedicated to my grandmother who recently passed away and who has always encouraged me with my academic pursuits

Table of Contents

List of Tables	9
List of Figures	11
Abstract.....	13
Acknowledgements.....	15
Chapter 1 Introduction	17
Chapter 2 E-Science and High Performance Computing	21
2.1 High Performance Computing.....	21
2.2 Computing Clusters	22
2.2.1 Sun Grid Engine	22
2.3 Grid Computing	23
2.4 Parametric Simulations	25
2.4.1 Nimrod.....	26
2.5 Large Scale E-Science and Grid Application Development	27
Chapter 3 Virtualisation	31
3.1 Virtual Machines	31
3.2 Platform Virtual Machines	32
3.3 Virtual Machine Images.....	35
3.4 Virtualisation in Grid Computing.....	36
3.4.1 Virtualisation in High Performance Computing	37
3.4.2 Virtual Machines as Infrastructure	38
3.4.3 Virtual Machines as Work Units	39
3.4.4 Orchestrating Virtual Machines across the Grid	40
3.5 Virtual Appliances	43
3.6 QEMU	44
Chapter 4 Architecture of Virtual Machine Work Units	45
4.1 Requirements for Supporting Virtual Machines in Grid Computing.....	45
4.2 Grid Architecture with Virtual Machine Work Units.....	46
4.3 Virtual Machine Work Unit Structure	47
4.4 Tailored Execution Environments within Virtual Machine Work Units	49
4.5 Typical Scenario Usage of Virtual Machine Work Units.....	50
Chapter 5 Design and Implementation of Virtual Machine Work Units and Supporting Utilities	53
5.1 Virtual Machine Work Unit	53
5.1.1 Base Image.....	54

5.1.2 Application Image	55
5.1.3 Output Image.....	56
5.1.4 Execution Environment	56
5.2 Creating Virtual Machine Work Units	59
5.2.1 Process.....	60
5.2.2 Design of the Virtual Machine Work Unit Packaging Utility	65
5.2.3 Virtual Machine Disk Database.....	67
5.2.4 Interface and Configurable Options	68
5.2.5 Virtual Machine Auto-Run Script.....	70
5.3 Launching Virtual Machine Work Units	71
5.4 Virtual Machine Work Unit Framework.....	74
5.4.1 Virtual Machine Disks	75
5.4.2 Virtual Machine Monitors	80
Chapter 6 Demonstration and Performance Considerations	83
6.1 Demonstration of Virtual Machine Work Units	84
6.1.1 Grid Application 1- Mandelbrot Set	84
6.1.2 Grid Application 2- Phaser	88
6.2 Virtual Machine Work Unit Metrics	92
6.2.1 Virtual Machine Work Unit Instantiation Performance Metrics.....	93
6.2.2 Performance of Virtualisation used by Virtual Machine Work Units.....	94
6.3 Virtual Machine Work Unit Feasibility	95
Chapter 7 Conclusion and Future Work	97
7.1 Review	97
7.2 Future Work	99
References	101
Appendix A – Virtual Machine Work Unit Packager Configuration Options	105
<i>Appendix B</i> – Virtual Machine Auto-Run Script Template Token Substitutions.....	108
Appendix C – Virtual Machine Work Unit Packager Utility Dependencies.....	109
Appendix D – Virtual Machine Work Unit Launcher Configuration Options.....	110
Appendix E – Virtual Machine Work Unit Launcher Utility Dependencies	112
Appendix F – Test Result Tables	113

List of Tables

Table 5.1: Schema of the virtual machine disk database	68
Table 7.1: Virtual Machine Work Unit start-up, application, and shutdown timings. Values are in seconds	113
Table 7.2: Size metrics of the Virtual Machine Work Unit. Values are in megabytes.....	113
Table 7.3: Performance results of Virtual Machine Work Units using the Mandelbrot grid application with different virtual machine execution types. Values are in seconds.....	113

List of Figures

Figure 3.1: Platform virtual machine implementation types	32
Figure 4.1: Traditional method of launching applications over the grid	46
Figure 4.2: Method of launching applications through Virtual Machine Work Units over the grid	47
Figure 4.3: Virtual Machine Work Unit structure	48
Figure 4.4: Base images do not necessarily need to be configured by e-scientists	50
Figure 5.1: The virtual machine disk components that make up a Virtual Machine Work Unit	54
Figure 5.2: The algorithm of the Virtual Machine Work Unit daemon	58
Figure 5.3: Overview of the packaging utility that allows the creation of Virtual Machine Work Units for e-scientists	60
Figure 5.4: The process of creating Virtual Machine Work Units.....	60
Figure 5.5: Design of the Virtual Machine Work Unit packager utility.....	65
Figure 5.6: Design of the virtual appliance class used by the packaging utility	66
Figure 5.7: Design of the application class used by the packaging utility	67
Figure 5.8: Design of the virtual machine disk database class used by packaging utility	68
Figure 5.9: Design of the Virtual Machine Work Unit launcher utility	71
Figure 5.10: Process of the launching utility used to execute Virtual Machine Work Units on a grid resource	73
Figure 5.11: Overview of the framework used by the packager and launcher utilities	74
Figure 5.12: Class diagram of the virtual machine disk classes	75
Figure 5.13: Support for virtual machine monitors	80
Figure 6.1: Visualisation of the Mandelbrot set	84
Figure 6.2: Configuration script for the Mandelbrot grid application	86
Figure 6.3: Base images that can be used to encapsulate the Mandelbrot grid application ..	87
Figure 6.4: Generated virtual machine auto-run script for the Mandelbrot Virtual Machine Work Unit.....	87
Figure 6.5: Configuration script for the Phaser grid application	89
Figure 6.6: Base images that can be used to encapsulate the Phaser grid application	90
Figure 6.7: Generated virtual machine auto-run script for the Phaser Virtual Machine Work Unit.....	91
Figure 6.8: Experiment file that is directed into the Phaser grid application within the virtual machine.....	91
Figure 6.9: Nimrod plan file for the Phaser experiment.....	92
Figure 6.10: Approximate Virtual Machine Work Unit running times	93
Figure 6.11: Size metrics of Virtual Machine Work Unit components	94
Figure 6.12: Mandelbrot grid application performance comparisons	95

Dynamic Generation and Execution of Virtual Machine Work Units on the Grid

Timothy H. Ward
Bachelor of Software Engineering (with Honours)
thwar1@student.monash.edu.au
Monash University, 2008

Supervisor: David Abramson
David.Abramson@infotech.monash.edu.au
Associate Supervisor: Wojtek Goscinski
Wojtek.Goscinski@infotech.monash.edu.au

Abstract

Grid computing offers significant promise as the next generation platform which will drive large-scale e-science. However, e-scientists are faced with challenging problems when developing and deploying grid applications. This includes heterogeneous nature of grid resources that constrain the e-scientist in developing applications designed for the grid rather than the grid providing the environment for the application.

Virtualisation offers great potential in solving this issue and a number of research projects have concentrated on applying platform virtual machines to alleviate these issues. Virtualisation in high-performance computing has become a viable option for encapsulating grid applications. However utilising such architectures still requires the e-scientist to have the skills and knowledge of setting up such environments.

This thesis proposes a flexible Virtual Machine Work Unit architecture that allows e-scientists to easily package their grid applications into a virtual machine that meets the requirements of their grid application. It also documents the utilities which have been developed to launch Virtual Machine Work Units on to existing grids with minimal or no modification to existing grid infrastructure. Virtual Machine Work Units are executed like traditional grid applications without the technical and organisational constraints found in existing grid infrastructures.

Dynamic Generation and Execution of Virtual Machine Work Units on the Grid

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Timothy H. Ward
October 31, 2008

Acknowledgements

I would like to thank my two supervisors David and Wojtek for their help and support, who put up with my lack of punctuality when it comes to providing drafts before submission deadlines, and providing the necessary guidance during the completion of this thesis. I would also like to thank my industry project team who has made this year an enjoyable one with the implementation of Marshall...

Timothy H. Ward

Monash University
October 2008

Chapter 1 Introduction

Grid computing has opened up possibilities for e-Scientists to conduct and collaborate on computer intensive experiments which would have once been infeasible[1]. High-performance computing experiments require access to large scale computing resources and data storage, and grid computing has made this possible. However, e-scientists are faced with challenging problems when developing and deploying grid applications.

Due to the nature of the computing landscape, grids commonly consist of heterogeneous resources; every resource on a grid can potentially have different architectural characteristics and software configurations. Grid resources are also constrained by organisational policies defined by the grid resource owner. For an e-Scientist to successfully utilise the full potential of a grid they must tailor their experiment to run on all or a subset of these resources across technical and organisational boundaries. The grid application is designed for the grid rather than the grid supporting the needs of the application. In most cases an e-Scientist may have some experience in software development. However, their main concern is in their field of research.

The process of developing and deploying software across a range of platforms, configurations and organisational boundaries is challenging for e-scientists[2, 3]. The development of grid applications is constrained to the grid resources on to which it is executed on. This can cause issues when the application developed is not compatible with available and existing grid infrastructures. This is common occurrence if the application has obscure requirements that cannot be tailored for by grid resources. The heterogeneous nature of grids also constrains supported applications to a subset of available resources limiting the potential for completely utilising the entire grid for experimentation. As grids continue to mature and be updated to support improvements, legacy applications may no longer be compatible with changes also causing difficulties in deploying grid applications.

One approach to reducing the effort required for developing grid applications is using virtualisation to abstract resource characteristics and allow e-scientists to define their own run-time environments for their grid applications[4]. Using this approach removes potential application development issues such as portability from the e-scientist's responsibility. This can be achieved by using platform virtual machines, which emulate a complete machine including its hardware, operating system, and software. These virtual machines can then be used to encapsulate the operations and data of the grid application, and allow e-scientists have complete control in defining the required execution environment to be used by their grid application.

Using virtual machines as work units to encapsulate grid applications provides a number of advantages over traditional grid application execution and deployment architectures.

- The architecture of the virtual machine can differ to the architecture of the underlying grid resource by using emulation at the cost of performance.
- The guest operating system of the virtual machine can differ to the host operating system of the underlying grid resource and can provide similar performance using virtualisation.
- The software stack within the virtual machine is completely controllable and can be tailored to meet the requirements of the grid application.

- The grid application is isolated from the grid resource allowing the greater trust of grid applications being launched across organisational boundaries.
- Deploying grid applications is limited to copying the virtual machine to the grid resource and executing the virtual machine on a virtual machine monitor.

However applying virtual machines to grid computing poses problems for e-Scientists as the configuration of such environments requires knowledge of operating systems concepts and system administration. Furthermore, the process of creating and configuring these environments can be tedious. Environments initially need to be configured with the base requirements for the application such as an operating system, software libraries, and other dependencies. The grid application then needs to be configured and installed within the environment. Experiment data then needs to be sourced and passed into the environment either from the experiment repository or being streamed from another grid application. The application then needs to be executed and monitored within the virtual machine. Once the experiment is completed the experiment results need to be extracted from the virtual machine and passed back to the experiment repository or passed on to another grid application. The environment then needs to be cleaned up to allow the releasing of the underlying grid resource.

This project aimed to look at how virtual machine can be automatically and dynamically created as work units and develop utilities to support this architecture. This included investigating the issues related with the development and deployment of grid applications on to virtual machines, and how the virtual machines can be deployed, maintained, and orchestrated across the grid. There by, simplifying the effort required by e-Scientists to conduct high-performance computing experiments on grid infrastructure as well as in the hope of a greater adoption of grid computing.

In response to the development and deployment issues faced by e-scientists, this thesis proposes a flexible grid virtual machine architecture that supports e-scientists to use virtual machine across existing grid infrastructures. This led to the design of Virtual Machine Work Units that allowed the flexibility required to dynamically generate and execute these grid application packages with requiring limited or no modification to existing grid infrastructure.

To demonstrate the feasibility and performance of the Virtual Machine Work Unit architecture a packaging utility that allows e-scientists to simply pass the grid application and environment requirements was implemented. The packaging utility would then return a Virtual Machine Work Unit that could be utilised on exiting grid infrastructure. This also included the ability for e-scientists to develop and tailor their own complex execution environments required by their grid applications. Supporting utilities were also implemented for providing the ability to pass instance-specific settings for grid applications encapsulated in Virtual Machine Work Units, and to launch Virtual Machine Work Units without access to virtual machine monitors directly installed on grid resources.

Applying this architecture an e-scientist is able to utilise the advantages of platform virtual machines in grid computing by using the supporting utilities implemented to easily generate Virtual Machine Work Units that support the required execution environment needed by their grid application. These can then be used on existing grid infrastructure by using the

implemented launcher utility that allows Virtual Machine Work Units to be launched on existing grid infrastructure with minimal or no modifications. The architecture also promotes the reuse of Virtual Machine Work Units, by reuse the individual components that make up the Virtual Machine Work Unit for other experiments and grid applications.

To greater understand what is required for incorporating virtualisation into grid computing; we will first look at grid computing and the challenges faced by e-scientists in developing grid applications (Section Chapter 2). This will then be followed by a technical review of virtualisation (Section Chapter 3), specifically examining virtualisation in high-performance computing, how it is applied in grid computing, and the concept of virtual appliances. Emerging architectures will be discussed with a concentration on platform virtual machines as work units.

The Virtual Machine Work Unit architecture is presented (Section Chapter 4), including the necessary infrastructure for supporting the dynamic generation and execution of Virtual Machine Work Units. The implementation of Virtual Machine Work Units, the protocols for tailored execution environments, and supporting packaging and launching utilities (Section Chapter 5) will be discussed in depth highlighting the advantages and disadvantages of design choices.

Demonstrations of packaging grid applications in Virtual Machine Work Units (Section 6) will be presented to show the necessary interaction of the e-scientist in creating and launching tailored environments. The performance metrics of generating Virtual Machine Work Unit environments will be analysed and a brief review of the performance of Virtual Machine Work Units on existing grid infrastructures (Section Chapter 6) will be discussed.

Chapter 2 E-Science and High Performance Computing

E-Science [5], also known as cyber-science, is the use of computing resources in aiding and supporting of complex experimentation. Nentwick[6] in his book about e-Science compares this against traditional science which he refers to as “science and research without the use of networked computers”. E-Science - the next evolution of science - exploits the power of computation and immense data sets and allows e-Scientists to conduct experiments that were once not possible. For example, Scientific fields in bioinformatics, social simulations, earth sciences, and particle physics can benefit from e-science to help deal with the large amounts of data and computational complexities [7].

E-Science incorporates the use of collaborative communication information technology in the sharing and distribution of scientific effort. Previously research and science was conducted by individuals with very little collaboration. However with the growth of information technology and the global community, science and research is now being conducted across many national borders and of people in many different fields. E-Science facilitates the sharing of resources, from scientific resources to computational resources, and instrumentation resources.

E-Science has been made possible with the use of computing technologies such as the Internet, networked computers, peer-to-peer computing, high-performance computing and distributed computing. Users of e-Science paradigms are generally referred to as e-scientists.

2.1 High Performance Computing

E-Science experiments can involve the processing large amounts of data which can be computationally expensive or in some cases data is not present and is strictly computationally expensive. Such computations, without exploiting parallelism, can take amounts of time that exceed the existence of the earth. However, when exploiting parallelism these computationally experiments can be computed within reasonable amounts of time. The paradigm of pushing computer performance is referred to as high-performance computing (HPC).

Kuck[8] argues the critical issue in high-performance computing directly related to design of software for exploiting the parallelism provided by modern computing architecture. As such different computing architectures have emerged as a result of computer scientists rethinking the design of software from sequential computation to parallel computation. Such computing infrastructures include supercomputers, cluster computing and grid computing.

Initially high-performance computing started with mainframes and single supercomputers that were often made up of many processors. This provided a centralised way of providing high-performance computing, though the cost of such computing hardware is expensive and generally outside the budget of most research initiatives. Even when such infrastructure was available, access to computing time was often restricted and under great demand. As a result less expensive high performance computing architectures came about such as cluster computing and grid computing.

2.2 Computing Clusters

Cluster computing was the next step in the evolution of high-performance computing. Instead of developing a single supercomputer, cluster computing uses multiple separate computers to provide the mechanisms needed for parallelism to fulfil the requirements of high-performance computing. This cluster of computers appears as a single logical computer and such has approximately the combined performance of all the computers in the cluster minus management overheads.

Sterling[9] defines a computer cluster as “any ensemble of independently operational computers integrated by means of an interconnection network and supporting user-accessible software for organising and controlling concurrent computing tasks that may cooperate on common application program or workload”.

The power of cluster computing can be shown by the significant occurrence of computing clusters in the top 500 fastest computers in the world[10]. The Beowulf cluster[11] is a prominent example of cluster computing. An implementation of networks of workstations[12] and parallel workstations, the Beowulf cluster moves away from traditional high-performance computing methods of using specialised computing infrastructure such as supercomputers with multiple processors and instead opting for the use of “commodity parts” available significantly reducing the costs. Combining a large number of workstations allows for the Beowulf cluster to provide infrastructure that supplied large computational power and data storage to be utilised by scientists for conducting earth and space scientific experiments and simulations.

However the costs of cluster computing are still an issue as dedicated computers are required for enabling the cluster and to truly reach the high-performance of modern e-Science requirements a large quantity of computers are required, which each in their own right take up physical space and energy. Due to nature of cluster computing, these dedicated workstations are homogeneous in nature as they often have very similar characteristics in terms of hardware and software. This allows easier development of applications for cluster computing and management of such computing environments. However the integration between the workstations is different in architecture than traditional multi-processor computation and often relies on computer communication techniques to ensure that the high-performance can be maintained[1].

2.2.1 Sun Grid Engine

Sun Microsystems provides its own open-source tools for the management of computational clusters, known as the Sun Grid Engine (SGE) [13]. The SGE provides a centralised tool for the management of resources within a computing cluster and controlling the distribution of jobs across the cluster (remote job execution system), matching user requirements with resources that meet those specifications[13]. The SGE allows the use of heterogeneous resources and such provides a powerful system for utilising unused computational resources across a computing cluster.

Submission of jobs within the SGE is a simplified process that allows users to concentrate on the implementation of applications rather than the requirements of deploying and executing the application over many resources. Using a command called “qsub” (queue

submit), the user passes the application to be executed across a single resource; this is added to the various queues offered by the specific computing cluster. The Sun Grid Engine then takes this executable and finds an available resource for it to be executed on. Feedback on the status of the job can be given by using the “qstat” (queue status) which returns the status for all queue submissions by the user. Each job is assigned a unique ID and application information such as the standard out and standard error streams are forwarded to the user to be used in debugging and/or retrieving results of the execution.

Remote job execution systems are well suited toward parametric experiments, as each job can represent a different parameterised instance of the grid application. This remote job execution also provides the necessary infrastructure for launching Virtual Machine Work Units, as these work units are generic computational jobs.

2.3 Grid Computing

Cluster computing provided the requirements of high-performance computing, however it required that resources were dedicated and homogeneous in nature. Furthermore with the presence of the Internet and the large quantity of computer networks available and cheap bandwidth; the utilisation of such heterogeneous and dynamic resources has been made possible with grid computing[14].

The concept of the computational grid was introduced as an analogy to the power grid of the 20th century[14]. Foster and Kesselman describe it in the terms of computational cycles being the same as electricity and such should be available as a universal service. The grid in which they refer to as in the terms of electricity, should provide a “reliable, low-cost access to a standardised service, with the result that power ... became universally available”[1]. This meaning in the terms of computation grids is a computing infrastructure that provides a service with standard interfaces, widely available, and inexpensive to use[1].

Foster and Kesselman define the following attributes those which are critical to the concept of the computational grid service and the adoption of grid computing and its viability in the future as a new computing paradigm [1]:

- Dependable – A guarantee that computational performance will be maintained and acceptable to the user.
- Consistent – Standardised interfaces and protocols to access the grid.
- Pervasive – A guarantee that the computation grid is always available regardless of location.
- Inexpensive – That access to computational resources is affordable.

Grid computing provides the premise of exploiting under utilised loosely-coupled resources, the increased use of parallel computing, the formation of virtual resources and virtual organisations, access to additional heterogeneous resources, distributed resource balancing, increased reliability with redundancy, and management across organizational boundaries[15]. Grid resources can include computational resources, storage resources, network resources, and instrumentation resources[16].

Grid computing middleware is still maturing and easing the issues deployment and management of computational grids. Grid computing in e-Science is being utilised in many

scientific projects. Undoubtedly the most recognised use of grid computing is SETI@home[17] and Genome@home[18] where both projects rally up public support for their projects, in computing signals for detecting the presence of intelligent life and unravelling the human genome respectively, by utilising the unused computational resources of their supporters.

The interactions of the e-scientist are important for the adoption of the grid. The logical process of interaction between the grid and an e-scientist (user) is usually as follows[19]:

1. User requiring grid access organises account creation and then ensures grid access is available by installing grid software to join their computing resource to the grid.
2. The user then proceeds to connect to the grid by using the software installed. The user is then usually required to authenticate using their previously created account.
3. Users then may query the grid to determine if there enough computing resources available for usage. If available the user then proceeds to submit their job on to the grid.
4. Users may need to specify data configurations for streaming into multiple jobs.
5. Once a job is submitted users may need to monitor the jobs as they execute to ensure completion.
6. If required users may need to reserve certain grid resources for use during their jobs.

To construct a grid requires the cooperation of all the resources within the grid requiring a set of procedures and protocols that define the grid architecture[20]. These procedures and protocols need to be defined for communication, computation, security, scheduling, and resource brokering. To orchestrate and facilitate these protocols and procedures implementations of frameworks have been developed to help define grid computing environments.

One such grid framework is known as Globus[21]. Foster and Kesselman define Globus as a low-level toolkit that provides “basic mechanisms such as communication, authentication, network information, and data access”[21]. The Globus Toolkit provides the needed foundation support by abstracting all the above mechanisms in to what Foster and Kesselman refer to as the Globus Metacomputing Abstract Machine[21]. This abstraction allows higher-level services that assist in developing and managing e-Scientist applications sit on top of this infrastructure using the features and mechanisms provided by this toolkit.

This abstraction is critical to adoption of grid computing, however with the recent popularity of web services, collaboration between major vendors and the Globus team have led to the creation of the Open Grid Services Architecture(OGSA)[22] in which the models and designs of grid architecture and web service architecture is being combined[20].

However, even with such infrastructure there are many challenges still left in grid computing. Due to the nature of grid computing, application development for this new computing paradigm will still prove difficult even with the advances made in distributed computing over the last few decades[1]. This is related to the heterogeneous makeup of the grid.

Managing and deploying grids also provide challenges as grid infrastructure needs to be ported and setup on computing resources. Resources need to be monitored and analysed to provide feedback to ensure that the performance of the grid is consistent. The nature of grid computing means that resources can be removed and added to the grid making grid computing a dynamic computing environment and may present issues to grid application development.

Another issue with grid architecture is that grid resources are used alongside their local users and such resource resources such as the CPU, memory, and storage are shared. This can lead to security issues if sensitive applications and data are run across the grid as these may be accessed without authorisation. Vice versa, grid applications running may in turn maliciously attack the grid resource. Such problems as these limit the potential of grid computing.

Grid applications can be developed using different paradigms for use in high performance computing. One approach is the creation of distributed applications that are executed on individual resources and communicate across the distributed system to each other during execution using parallel programming techniques. This can be effective for certain experiment domains being solved by grid applications; however it requires the developer to create a distributed application that handles the management of resources, the fault-tolerance of resource failures, and the appropriate synchronisation between messages. Implementations such as the message passing interface (MPI) can be used in the development of distributed applications to simplify this[23]. This paradigm defines a set of communication protocols for communicating with parallel entities and is designed to be language independent, scalable, portable, and high-performance[23]. Support for MPI is implemented in most programming languages through APIs.

The second approach is using work units (jobs) that execute independently on a subset of data. These work units are launched in parallel across grid resources and once all completed the results can be combined or presented separately. The advantages of this paradigm are that the creation of these applications is simple and straightforward requiring no modification when being executed across distributed systems.

2.4 Parametric Simulations

One approach to high-performance grid application development is the creation of work units to be executed across grid resources. These work units represent a body of execution that can be applied to a subset of data, and once executing in parallel alongside other work units, provides high-performance computing for tackling computationally intensive problems. One such use of this paradigm is in parametric simulations.

Parametric simulations involve the execution of a computational simulation in which parameters are provided to the simulation which in turn determine the outcome of the simulation. These outcomes can be used to determine various scenarios based on the initial parameters. This technique is widely used in various research methods such as simulating and testing aircraft designs, modelling climate change based on certain hypotheses, designing and synthesising crystalline structures, and other modelling simulations that are used to represent real-world problems.

Conducting such simulations is computationally expensive and often multiple simulations are needed to be run using different parameters to provide useful data on different hypotheses. Computational grids provide the necessary infrastructure to complete these multiple simulations within sufficient time constraints. Multiple simulations can be executed concurrently across grid resources, each with their own distinct parameters, and once completed results can be combined or presented separately.

2.4.1 Nimrod

Developing parametric simulation grid applications still poses some problems for e-scientists. Even though the simulation code remains untouched, supporting infrastructure required for launching simulations across the grid can still be overwhelming for e-scientists to manage. E-scientists still need to create each instance of the parameters for the simulation, then launch each simulation on to the grid through a job submission utility, and then develop some method of fault tolerance in case a particular job execution unexpectedly terminates. This requires a lot of attention of the e-Scientist to the details and implementation of grid infrastructure and removes the e-Scientist from the domain of their experiment.

In response to these issues, grid middleware tools were developed such as Nimrod[24] to help e-scientists concentrate on their experiments. Nimrod provides the necessary automated management infrastructure required by e-Scientists for conducting parametric simulations. It combines the advantages of distributed application and remote job execution methods by providing an interface tailored to scientific simulations provided by tailored distributed applications and the power of launching experiments across the grid abstracting the infrastructure of job and resource management as provided in remote job execution systems[24]. Later versions of Nimrod have been designed to utilise the Globus Toolkit[25].

E-Scientists can utilise Nimrod by logging on to an externally hosted web-portal and/or using client-side software installed on their desktops. E-scientists are then presented with a management console that allows them to create experiments, view the status of current experiments, and the resources available to the experiments.

An experiment within Nimrod provides an interface for the e-Scientist to design and execute their parametric simulations. Nimrod provides the ability to define parameters including static and dynamic parameters for several different data types. Nimrod takes the cross product of these parameters and creates jobs for each set of parameters. Experiments are broken down into different tasks: experiment pre-processing, execution pre-processing, execution, execution post-processing, and experiment post-processing[26]. The e-scientist can use these various stages to copy data to and from the parametric simulation, and use experiment processing stages to split or collate data and results respectively.

The jobs generated by the Nimrod experiment are then launched across various grid resources. E-scientists can selectively choose which resource clusters within the grid they wish to utilise. Cost and time constraints can also be supplied and defined to allow Nimrod to determine the best schedule to ensure that cost and time constraints are met. This allows experiments to make use of more and expensive resources when time constraints are

crucial for experiments. Nimrod also provides an interface for e-scientists to monitor the status of their experiments, including determining which jobs are currently executing and provides an interface to launch selected jobs ahead of initial scheduling.

2.5 Large Scale E-Science and Grid Application Development

Grid computing has opened up possibilities for e-Scientists to conduct and collaborate on computer intensive experiments which would have once been infeasible. The next generation of large scale experiments for E-Science requires access to large scale computing resources and data storage. High-performance computing experiments can now be run without requiring a dedicated super-computer.

E-Science infrastructure now extends to providing users with science portals for easy access to such infrastructure, the ability to use distributed computing to allow computational experiments to be computed at high-performance, large-scale data analysis using the distributed storage, integration of other scientific resources such as radio and optical telescope data, and the distribution of collaborative work in the scientific community[14].

Foster and Kesselman[1] define the applications of a grid into five categories: distributed supercomputing, high-throughput computing, on-demand computing, data-intensive computing, and collaborative computing. These categories are core to large scale e-Science.

Due to the nature of the computing landscape, grids commonly consist of heterogeneous resources; every resource on a grid can potentially have different physical characteristics and a different configuration. For an e-scientist to successfully use the full potential of a grid they must tailor their experiment to run on all or a subset of these resources. In most cases an e-Scientist may have some experience in software development. However, their main concern is in their field of research. For e-scientists, the process of developing and deploying software across a range of platforms, configurations and organisational boundaries is challenging[2, 3].

Traditional software life-cycles follow a development, deployment, testing and debugging, and execution and this applicable to the development of grid software[27]. The two major challenges in grid computing are development and deployment of grid applications.

Development of grid applications requires a way of implementing software on many different platforms and computing architectures. This has been made possible by using interpreted and application runtime architecture; however use of such languages may not be suitable for some cases e-scientists as high-performance computing may need the performance of native applications. However, there has been significant advances in this area[28]. Unfortunately this still forces the e-scientist to develop an application for the grid infrastructure rather than the grid infrastructure being designed for the e-scientist's application. There are also other issues plaguing the development of grid applications such as platform incompatibilities, grid utilisation, support for legacy applications, obscure requirements, and the potential lack of backward compatibility with grid resource updates.

E-scientists who have developed grid computing applications though do not have access to the required infrastructure to support their implementation. This can be frustrating for e-

scientists who approach their problem directly and then find their work has been futile as it cannot be supported or executed on accessible grid infrastructure.

Grid resources available to the e-scientist may be limited to distinct subsets based on provided architecture of the grid resources. The e-scientist's application may be limited to a single subset of the available grid resources and does not provide the complete utilisation of the grid needed for the high performance requirements of their application.

Testing and debugging of grid software poses some issues as grid software behaviour may be dependent on specific grid resources. Reporting mechanisms can be used but still require investigative skills on part of the developer for the debugging of software. Execution requires the grid software to be scheduled and managed across the grid[27]. Unfortunately, during this execution grid software could fail and such software requires recovery mechanisms.

Support for developed for new and existing grid applications may not be provided by newer or existing grid infrastructures available to e-scientists and can prevent the utilisation of a proven method of experimentation. Support for legacy grid applications is crucial to ensuring e-scientists can concentrate and continue on research methods, however it is also critical to improve existing grid infrastructure to harbour and support future technologies and performance enhancements that provide greater benefit for newer grid applications.

Grid applications developed by e-scientists may be unique and different to existing grid application implementations and such may have obscure requirements necessary to the execution of the application. Existing grid infrastructures may conflict with such implementations and control of the grid resources may be limited and such provide no way of meeting the obscure requirements.

Developed grid applications being deployed and executed on existing grid infrastructure may face potential issues when changes or updates are made to the underlying systems of each of the grid resources. Grid resources may require these updates to provide the security and reliability needed, however incompatibilities may provide headaches to e-scientists in their research.

Deployment of grid applications requires a method of distribution across the grid and then deploying grid software to each grid resource for execution. Applications may be complex in nature and may require other software dependencies. Applications may also be required to be redeployed to grid resources as updates are made to the application. As mentioned specific grid resources cannot be assumed to exist as they may be added and removed at any point during an application's deployment and execution. This can pose issues to developers unless the abstraction of such resources is utilised. Grid applications with specific requirements can only be deployed to grid resources that support these requirements. These requirements are usually specified by the developer, however often need to be approved and installed by grid administrators. This delays the utilisation of the grid for scientific experimentation and can often deter the development of complex grid applications required by e-scientists.

In response to such issues in software development for grid software specifically development and deployment, several implementations of frameworks and architectures have been created; Abramson defines this as “Upper Middleware”[27].

One such grid framework that implements this upper middleware layer is the “Infrastructure for the Deployment of e-Science Applications” (IDEA)[29]. IDEA provides tools for managing deployment across grid heterogeneously through DistAnt[2, 3]. It also provides an application-runtime environment and automatic deployment tool for grid software. Another architecture for the creation and deployment of grid software is the Grid Application Development Software (GrADS) Software Architecture[30].

One approach to reducing the effort required for developing and deploying grid applications is using virtualisation and emulation to abstract resource characteristics and allow e-scientists to define their own run-time environments for their grid applications[4].

Chapter 3 Virtualisation

Virtualisation is a technique of abstracting the underlying physical computing resources into logical computing resources. These virtualised resources are accessed by a well-defined interface that maps interactions to the underlying implementation of the physical computing resource[31]. Virtualisation is the partitioning or consolidation of physical and no-existent resources into well-defined logical resources. That is virtualisation can allow multiple physical computing resources to be presented as a single logical computing resource, the virtualisation of a single physical computing resource into multiple logical computing resources, the simplification of a physical computing resource into a logical resource using encapsulation, and the emulations of a logical computing resource that has no matching physical underlying computing resource.

Virtualisation of resources can be extended to all aspects of computing. Resources such as CPU's, memory, disk-storage, external-media drives, network devices, graphic hardware, keyboards, etc can all be virtualised using the above techniques. Virtualisation can even be extended to virtualising entire computing environments, external computing equipment, and entire networks.

The roots of virtualisation first started around the 1960s in its application of mainframe computing where single hardware resources were shared by multiple users[32]. In an attempt to segregate different user and their interactions with the mainframe, virtualisation was used to present a separate logical computing environment for each user. This prevented users from interrupting and interfering with each other and provided a more reliable and recoverable computing environment. However, after the lapse in mainframe computing to desktop computing, the use and implementations of virtualisation declined, and it was not till recently that virtualisation has again become popular in research and in industry.

3.1 Virtual Machines

The use of virtualising an entire computing environment has allowed for the concept of virtual machines. These virtual machines can emulate a fully functional computing environment including hardware, operating system, and applications. However, some virtual machine implementations do not virtualise all these aspects. These virtual machines generally sit on top of a physical computing system and/or its operating system.

Virtual machines can be broken down into two main categories, system virtual machines and process virtual machines. Each provides similar advantages and disadvantages. System virtual machines emulate from the hardware instruction architecture level, whereas process virtual machines emulate from the process level.

Process or application level virtual machines are emulating the running of a process within the computing environment. This kind of virtualisation technique has been adopted by high-level languages that are interpreted and keep a separate machine state for each execution. Process virtual machines are used to provide application portability by using virtual machines to allow platform independence. Examples of popular application virtual machines include the Sun Java programming language that runs on Java Virtual Machines (JVM) and

the Microsoft .NET programming language family that runs on Common Language Runtime (CLR) virtual machine.

3.2 Platform Virtual Machines

Platform virtual machines abstract the entire computing resource by virtualising the underlying computer hardware thus emulating a complete computing environment in which the user actions and/or executions will not directly affect the underlying resource[31]. Virtual machines can either be emulated or virtualised, that is the computing environment is interpreted during its execution or executes non-sensitive instructions natively on the underlying hardware respectively. A variant on virtualised virtual machines is Para-virtualisation where the virtual machine's operating system is modified for virtualisation purposes.

Virtual machines are controlled by what is known as a hypervisor or virtual machine monitor (VMM). The responsibility is to manage the virtual machines by controlling computation, memory access, and other virtualised resources. The VMM implementation is relatively small compared to operating systems and in most cases is used only to catch sensitive instructions.

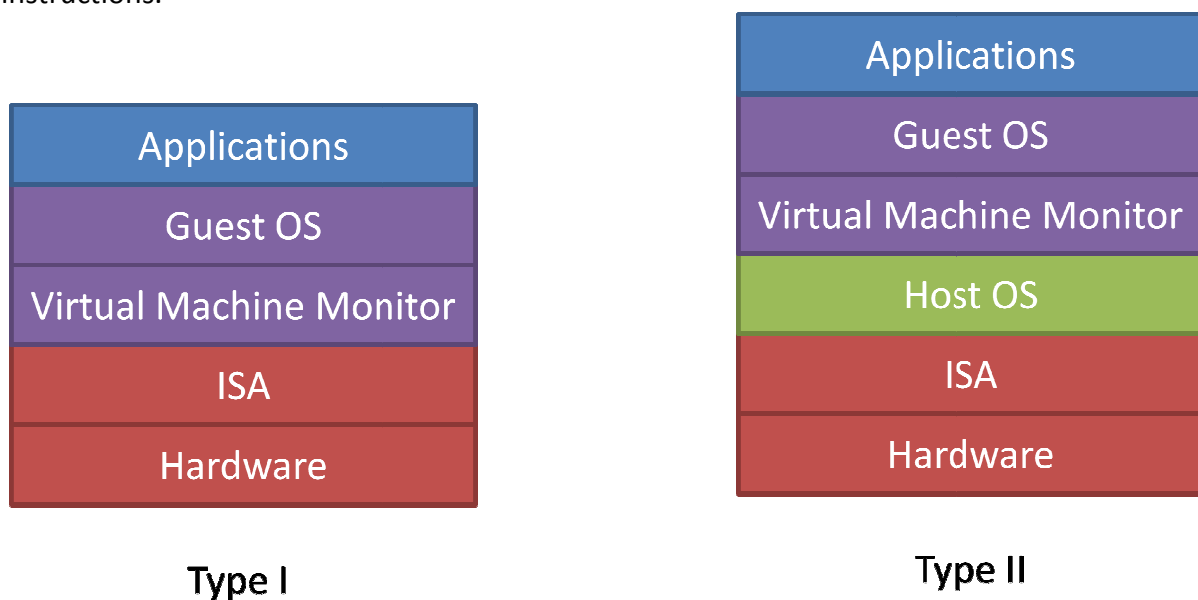


Figure 3.1: Platform virtual machine implementation types

The virtual machine monitor can sit on top of the existing host operating system and/or directly on top of the hardware. These implementations are known as type II and type I respectively and can be seen in Figure 3.1. The host operating system is the physical machines run-time environment whereas the guest operating system is the run-time environment run on the virtual machine.

Virtual machines allow the partitioning of a physical machine into multiple concurrent virtual machines. This is achieved by partitioning the underlying resources which are used by the virtual machine. Each virtual machine uses a subset of resources of the physical machine by allocating or sharing them between other virtual machines. Each virtual machine can run different operating systems and even emulate different computing platforms.

Likewise, virtual machines allow the opposite of partitioning. Virtual machines can be used for consolidating multiple physical resources into a single logical resource. When a virtual machine make a call to access the logical resource, underlying mechanisms determine what physical resource(s) need to be accessed. This could also be extended to the ability of combining multiple physical machines into a single virtual machine.

When multiple virtual machines are shared on the same host, resources are normally shared between executing virtual machines. However, each virtual machine should not interfere with the performance or operations of a concurrently running virtual machine. This is enforced by the virtual machine monitor that ensures that the virtual machines stay within their boundaries, and is inherited from the techniques pioneered in operating systems with multi-programming and processes. If the virtual machine is sharing with a host operating system the virtual machine monitor implements the isolation layer between the two. That is the host operating system sees the virtual machine monitor and virtual machines as a single process and treats it accordingly, though the virtual machine monitor, depending on implementation, may be integrated into the operating system for performance reasons. This separation is known as isolation and encapsulates the entire computing environment into the virtual machine, however even with such techniques of isolation mechanisms this isolation is only encouraged and not guaranteed[33].

Virtual machines can be configured with certain resource allocation limits. This prevents virtual machines from dominating a certain host machine. Due to the dynamic nature of virtual machines they can also be reconfigured while being executed. This allows virtual machines with increased workloads to adapt and virtual machines that are idling to be slowly decommissioned unless woken.

The implementation of virtual machines normally incorporates a virtual machine monitor and the virtual machine which are simply a process and a file respectively. The virtual machine file is known as the virtual machine image. This simplistic but powerful abstraction allows the exploitation of virtual machine states. This allows virtual machines to be paused and restarted at any execution point. From booting to application execution, virtual machines states can be created. Check-pointing of such states allows easy recovery if there are any issues within the virtual machine.

Another advantage of treating virtual machines as files is the automation that can be incorporated into the execution across multiple hosts. If a host machine is under load-pressure or ceases to be available, virtual machines can be migrated across networks to new hosts without interrupting the sequential execution of the virtual machine. In some implementations, live migrations can occur meaning that virtual machines are still executing even though the execution location of the virtual machine is changing. This is another recovery advantage of using virtual machines.

Automation is also provided in the sense that virtual machines can be dynamically reconfigured. The ability to reconfigure virtual machine resource usage and the ability to move virtual machines across multiple hosts allows the fine-grained control of physical resources. The communication and coordination between such hosting environments can ensure a high-performance and reliable computing environment.

The creation of virtual machines can also be automated due to the implementation of virtual machines as files. These environments can be created and cloned as required without the process of reproducing installations of software including the operating system.

Another important property of virtual machines is the portability provided making the virtual machines hardware independent. Like application virtual machines where the code of the application can be ported without changes to other machines by simply running the code; virtual machine images provide the same portability and such the same virtual machine image can be executed on any machine with the same virtual machine monitor. In response to such capabilities and the many implementations of virtual machine monitors, attempts at creating an open standard for virtual machine images is being lobbied. Virtual machine images will be discussed in more detail later in Section 3.3.

Finally another important attribute of virtual machines is their ability to provide legacy emulation for computing hardware that does not exist or is different to the underlying host's computer architecture.

Due to the many attributes of virtual machines, this technology has been applied to many areas such as server consolidation, development, malware analysis, trusted computing, and virtual applications.

There are many different implementations of virtual machine monitors. VMWare a leader in commercial virtualisation products implement its own virtual machine monitor[34]. VMWare implements the virtual machine monitor in type I and type II implementations known as VMWare GSX and ESX respectively. Their type I virtual machine monitor is based on a slimmed down Linux kernel. The virtual machine monitor is only implemented for the x86 architecture and likewise can only virtualise x86 virtual machines. Virtual machine images are stored in a file format known as Virtual Machine Disk Format (VMDK).

Xen[35] is another popular commercial virtual machine monitor. However, unlike the above two virtual machine monitors mentioned above it is only a type I virtual machine monitor implementation. The performance of Xen is one of the major driving forces behind its popularity.

Another virtual machine monitor worth mentioning but not in detail is the implementation of virtualisation in Microsoft's new Windows server product Server 2008[36]. This is re-entrance from Microsoft in the commercial server virtualisation market, however unlike other virtual machine monitors; this will be directly integrated into the operating system.

QEMU[37] is a type II virtual machine emulator written by Fabrice Bellard. QEMU provides support for a number of computing architectures through emulation and provides virtualisation through use of the KQEMU device driver. More information on QEMU can be found in Section 3.6.

3.3 Virtual Machine Images

As mentioned previously, virtual machine images represent the virtual machine and its state. Virtual machine images are host machine files that store this representation. The advantage of storing virtual machines as files is that they can be accessed and operated on the same fashion as any other file. In most cases a virtual machine image is simply the disk representation (partitions, boot sector, and file-systems) of the virtual machine.

Virtual machine images can be implemented to support dynamic growth based on disk usage or completely allocated at the creation of the virtual machine. Dynamic growth ensures disk space of the underlying host is only utilised if needed, however this comes at a cost of performance as space is allocated when the virtual machine requires additional space.

This is often exploited for allowing host-to-guest communication and guest-to-host communication. Meaning the host can access the virtual machines files, however this is dependent on the implementation and in some cases accessing the virtual machine image during its execution may inherently cause some damage.

Unfortunately there are many different implementations of virtual machine monitors and such many different file formats that represent virtual machine images. In response, attempts have been made to lobby the creation and adoption of an open virtual machine image standard. There have been many submissions for a standard, though the most likely contender is VMWare's Virtual Machine Disk Format (VMDK)[38].

The VMDK format supports dynamic growth or complete allocation of the virtual machines image. The format also extends to supporting a single file representing the virtual machine image and/or multiple files that combine to form the virtual machine image. These multiple files are linked together as a chain. Each link in the chain is made up multiple elements referred to as extents. The overall structure of the VMDK format includes a header where information such as versioning, machine identification, linking information for multiple files, creation method, and other information. The rest of the file body contains the data for the extents which represent the virtual disks. More detailed information can be found in the VMDK specification[39].

The advantage of having a single open virtual machine image standard is that a virtual machine is independent of the underlying virtual machine monitor and hence increasing the portability of such virtual machines over many host machines with different virtual machine monitors.

Some disadvantages may be the limitations of the specification, however due to the nature of virtual machines; further meta-data information could be encoded within the virtual machine's file system.

However until an open standard is adopted by major virtualisation vendors this may cause limitations. Fortunately there have been developments of creating virtual machine image translators that convert one virtual machine file format in to another. For example, in QEMU, the `qemu-img create` program provided alongside QEMU allows the conversion of

VMWare formats into the various file formats supported by QEMU[40]. However this translation is not reversible at this current stage.

Because virtual machine images represent hard disks and separate file systems, these files can be mounted like any other hard-drive and their file systems accessed. For example, in QEMU, the RAW image can be mounted using a loopback device and can be written and read from like any other device. VMWare provides a tool, called VMWare Disk Mount Utility[41], that can be installed to allow the mounting of the VMDK images. However, even with the mounting of disk images, the underlying host must still be able to interpret the virtual machine file-system.

The nature of files allows the use of snapshots. Snapshots represent the state of a virtual machine at any point during execution. These snapshots are normally stored as the changes from the original virtual machine image and/or the last snapshot taken. In the example of migration between hosts, the original virtual machine image and its subsequent snapshot files can be transferred across to the new host and the virtual machine can continue execution from where it last executed. Snapshot support however is dependent on the file format and virtual machine monitor that is being used.

3.4 Virtualisation in Grid Computing

Attempts have been made to standardise and make grid computing more accessible[21, 42]. However, even with such toolkits and standards, implementing infrastructure and developing for grid computing still remains a challenge[27, 29]. In response, virtualisation is being applied as a solution to this problem[4]. Virtual machines have been successfully applied to grid computing, using both application level virtualisation and platform level virtualisation [4, 43-57].

Application virtual machines that make use of .Net/Java virtual machine technologies are already implemented as middleware across different grid implementations[44]. These implementations allow e-Scientists to develop portable applications which can be executed across a range of environments. In some cases these environments support legacy code to a certain degree[44]. However, they do not give complete control for the e-Scientist to completely specify their experiments run-time environment; this includes controlling the underlying operating system and other legacy application dependencies.

Platform virtual machines abstract the entire computing resource by virtualising the underlying computer hardware thus emulating a complete computing environment in which the user actions and/or executions will not directly affect the underlying resource[31]. Platform virtual machines provide isolation, legacy-support, administrator privileges, resource control, and environment recovery[4]. Combined, these characteristics have the potential to provide a high level of control when conducting experiments in a grid environment. Furthermore, recent advances in virtualisation and virtual machines has led to performance overheads being dramatically decreased and has made it feasible to apply virtual machines to high-performance computing[58]. This is covered in more detail later on.

The integration of grid computing middleware with platform virtual machines has led to two major architectures. The first approach has led to the placing of grid middleware into the

virtual machine. This approach allows grids to be implemented and deployed by running these virtual machines on the grid resources[46]. The second approach is using existing grid infrastructure and using virtual machines as a work units for the execution of applications [43, 50, 51]. In this case the grid middleware is used for supporting the virtual machine deployment and execution, though it should be noted that the first approach can be used in conjunction with this method.

3.4.1 Virtualisation in High Performance Computing

High performance computing requires the performance of applications to be optimal and exploit parallelism. However, the effort required to implement high performance applications normally requires development of applications in low-level languages that are designed and configured to be optimally run natively on a designated machine.

The use of virtualisation leads to overheads that reduce performance. These can be accounted to the translation of instructions, the intercepting of sensitive instructions, file system access for reading and writing to the virtual machine image, network virtualisation communication, and subset of computing resources.

Recent advances in virtualisation however have now made it possible for high-performance computing to be considered for implementation on virtual machine implementations. Though for this to be adopted, users have to be assured that the performance of using such techniques will not hamper the performance of computation. As such, research has been conducted on the performance of virtual machines[4, 59, 60] as well as there feasibility for high-performance computing[58].

Macdonnell and Lu[58] in their performance analysis of virtual machines for high-performance computing used the popular VMWare GSX virtual machine monitor. They used scientific applications to ensure the verification of their results; BLAST, HMMer, and GROMACS[58]. The host hardware used the following specifications; dual opteron @ 2.2ghz, 4gb memory, 250g hard-disk and running Linux. Each virtual machine was allocated 2GB memory. There tests were aimed at finding how these machines performed under stringent I/O activity and computational performance for high-performance computing. Macdonnell and Lu in their results concluded that the overhead of using virtual machines for computational activities involved an overhead of 6%, while for I/O intensive an overhead of 9.7% was observed[58].

Application virtual machines have been developed for high-performance computing purposes. One such example is Motor[44]. Motor takes advantages of virtualisation and modifies an existing virtual machine, the Common Runtime Infrastructure (CRI). The modifications incorporate features needed in high-performance computing such as high performance message passing interface (MPI). Other modifications include the memory management to handle the inclusion of the MPI modifications. Results published indicate that the performance of such virtualisation techniques, however less than natively run applications, provide very good performance given the advantages of using virtualisation[44].

Providing the performance of virtual machines continually improves, the advantages of using virtualisation outweigh the overheads in performance for high-performance computing and as such should be utilised by grid computing.

3.4.2 Virtual Machines as Infrastructure

Virtual machines can be effectively used as the platform for supporting grid infrastructure. This is following the traditional steps of virtual appliances[61], where virtual machines are used to distribute software; in this case the appliance is the grid middleware.

This architecture allows users to have access to a uniform set of resources. That is, all grid virtual machines can be of the same architecture and hence requires less effort when developing and deploying grid infrastructure.

The portability of virtual machine images means that grids can be effectively implemented by distributing the virtual machine image. As long as the underlying resource has a virtual machine monitor installed, the resource can be added to the grid resources simply by instantiating the virtual machine.

Isolation between the virtual machine and host machine means that any grid infrastructure applications and applications running within this grid are kept separate from the underlying resources. Any faults within the grid resource will not propagate to the underlying host machine and such reassures grid users that grid computing is safe. Likewise, sensitive data and operations are also protected in a sense.

Reliability in the grid can also be assured by using virtual machines; if a resource goes down it can easily be recovered by copying the grid virtual machine image.

Grid Appliances[46] is one such example of using virtual machines for grid infrastructure. Users can join the grid by downloading the virtual machine image provided by Grid Appliances. Support for multiple virtual machine monitors is made possible by having different virtual machine image formats; for now it is VMDK and QCOW2. Once a user starts up a virtual machine, the modified guest operating system based on Debian Linux, configures network access which uses a peer-to-peer virtual network. This virtual network uses private IP addresses referred to as IPOP[62]. IPOP provides a way of distributing IP addresses without a centralised server. Inside the guest operating system, Condor is used for job submissions that run within the Debian operating system. Grid Appliances uses SAMBA within the guest operating system to setup a network share for the user's host machine. This allows users to copy in required data and applications. Users may also have root access to the virtual machine using SUDO provided within most Linux distributions. Grid Appliances provides a test pool of infrastructure of approximately around 500 nodes for testing grid job submissions and infrastructure and is distributed across the world.

However, unfortunately this only solves some issues for e-Scientists. Even though the grid resources are now homogeneous in nature, e-Scientists still need to develop applications and deploy dependencies to each grid resource. This restricts the set of resources they have access to. Likewise they do not completely control or able to completely define their execution environment and such can place restrictions when developing experiments.

3.4.3 Virtual Machines as Work Units

Virtual machines allow the encapsulation of data and operations and can be easily deployed to virtual machine monitors for execution. As such the use of virtual machines as work units allows e-Scientists to define their own run-time environment for an experiment application[4]. Virtual machine images are submitted as jobs rather than the applications and can be incorporated remote job submissions and/or into scientific workflow systems. Using this approach removes potential application development issues such as portability from the e-Scientist's responsibility. This can be achieved by using platform virtual machines as they emulate a complete machine including its hardware, operating system, and software. However this method still poses some problems for e-Scientists as the configuration of such environments can be time consuming and requires knowledge of operating systems concepts and system administration.

The process of creating and configuring these environments is tedious. Environments initially need to be configured with the base requirements for the experiment such as an operating system, software libraries, and other application dependencies. The experiment application then needs to be configured and installed within the environment. Experiment data then needs to be sourced and passed into the environment either from the experiment repository and/or being streamed from another experiment application. The experiment application then needs to be executed within the environment and be monitored to ensure that progress is made. Once the experiment is completed the experiment results need to be passed back to the experiment repository and/or passed on to another experiment application. The environment then needs to be cleaned up to allow the releasing of the underlying grid resource. These experiment applications are usually incorporated into a scientific workflow and as such this process of configuring environments needs to be repeated multiple times.

Virtual machines as work units are often referred to as sandboxes as they allow users to customise the execution environment without compromising the resource[52]. Some techniques have been developed so that the virtualisation level is abstracted and the platform, process, and other virtualisation levels are not specific to a particular environment; rather the environment is a dynamic virtual environment[47] or virtual workspace[48].

Santhanam et al[51] defined virtual machine sandboxes into four categories. The first category was a definition of virtual machines for infrastructure. The other three categories were more inline of the concept of virtual machines as work units. Work units may be entirely encapsulated without network access for execution, and would be setup with data leading up to execution.

Work by Adabala et al. [43], using the original findings by Figueiredo et al[4], led to the creation of a grid architecture that incorporated virtualisation. This virtual computing grid was referred to the In-VIGO (Virtualisation Information Grid Organisation) system. The system enables multiple application instances to be executed across virtualised and physical grid resources. In-Vigo also incorporates virtual file-systems, virtual machines, virtual applications, virtual networks, and virtual user interfaces[43]. To facilitate the creation and

deployment of virtual machines the In-Vigo system uses another grid infrastructure tool known as VMPlants[50].

Another implementation of virtualisation in grid architecture is an attempt to build on existing Globus architecture. In their work known as Virtual Workspaces, Keahey et al. [47-49] push forward the idea of virtual workspaces or dynamic virtual environments. These virtual workspaces represent the execution environments presented and used by e-Scientists. Users can “negotiate the creation of a new execution environments and system administrators to specify policies that govern there use and monitor there usage”[48]. There approach differs that instead of mapping jobs to resources, users now map jobs are mapped to workspaces. They also argue the need for abstractions of virtualisation techniques so not to restrict the use virtual workspaces.

3.4.4 Orchestrating Virtual Machines across the Grid

Using virtual machines as work units requires orchestration of various steps in the creation and configuration, deployment, and execution of such environments on the grid. Most of these steps would be outside the ability and effort for e-Scientists; however through automation most of these steps can be simplified and abstracted.

3.4.4.1 Creation and Configuration

The first step in an e-Scientist describing their execution environment is the ability to define the requirements of the environment. Customisation of such environments is generally through a specification which is provided by the user. These may be passed to a service for auto-configuration [43, 50]. VMPlant is one example of a virtual machine factory that is responsible for creating virtual machines based on a configuration file supplied by the user.

Configurations provide the ability to specify virtual machine specification. This includes the computational requirements, memory requirements, storage requirements, and any other device settings.

Configurations can be represented in a number of ways. VMPlants for examples uses directed acrylic graphs to represent the configuration and installation of software on to virtual machines. XML schemas provide another convenient way of specifying requirements. Other approaches include using Java or similar object oriented programming languages to specify requirements[61]. OOP language is used to allow inheritance in describing virtual machines.

This represents the basic description of a virtual machine; however more is needed in setting up the actual execution environment. Once submitted to a virtual machine image creation program, the virtual machine can now be executed. Users could use a local virtual machine monitor to launch this virtual machine and would be presented with interface to use this machine[4]. This could be through using VNC or other similar methods of terminal computing.

Because the e-Scientist has complete control of their Virtual Machine Work Unit, they have administrator access rights, and can tailor there execution environment as needed. This may is extremely powerful, however e-Scientists may not have the skills required to setup such

an execution environment. Other approaches for users to configure their virtual environment could be through using software installation approaches. Further information provided by the configuration could be used to specify the operating system, required network settings, application and application library dependencies, and user folder locations.

Given a configuration file, the virtual machine needs to be configured with the requirements. Due to the likelihood of many configurations sharing the same software requirements, e.g. operating system, one approach is to allow the user to make use of an existing bare virtual machine image with an operating system already installed. VMPlant uses this method for the initial building block of configuration. From this base image, VMPlant then mounts CD-ROM ISO images and uses the auto-run to launch scripts for the remaining configuration.

Other approaches include allowing virtual machines to be represented with multiple virtual disks, where each disk represents a different component of the virtual machine (e.g. operating system, software, libraries, etc). These are then combined to form the specified virtual machine. Wolinsky et al. [52] describes this approach using a file system known as UnionFS that allows the combination of multiple file systems. More direct methods include accessing the virtual machine image directly and installing the software from the host machine.

3.4.4.2 Deploying

Once a virtual machine environment has been configured and setup, depending on the workflow specified, the virtual machine has to be sent to the grid resource for execution. This virtual machine may be potentially cloned and sent to multiple grid resources.

Virtual machine images can be quite large depending on the execution environment tailored and can contain a lot of dark storage; that is storage that is not being utilised. This means that large file transfers are required when sending virtual machines. Using file transfer methods such as “on-demand” access can potentially be used to increase performance[58]. In the case if virtual machines are broken into multiple components (operating system, software, etc), then most likely the operating system components will be shared among multiple virtual machines and experiments and such can be cached locally[52].

Depending on the grid infrastructure, virtual machine monitors may not be installed on grid resources. In this case the virtual machine monitor needs to be sent along side the virtual machine image, or the virtual machine monitor needs to be installed onto the grid resource. If the virtual machine monitor already exists on the grid resource, then the correct virtual machine image format needs to be sent. This may require the original image being transformed into this new format. However, the virtual machine monitor could be sent explicitly regardless of the presence of an existing virtual machine monitor. This could be in the form of a virtual machine monitor like QEMU that can be executed as a process without requiring administrator access on the grid resource.

The applications within virtual machines will require storage access for the passing of input and output, this may be streamed in from another virtual machine, or accessing a data store.

One method is to copy the data directly into the virtual machine image when it is being initially configured. Other methods could include using network file systems and/or network communication protocols. Streaming between virtual machines may be incorporated into the workflow.

3.4.4.3 Executing

Once a virtual machine is received by the grid resource, it must be started. The starting of the virtual machine must ensure that the application is started. This can be managed by ensured by having a service within the virtual machine responsible for starting the application. Other approaches include using start up scripts to launch applications. Virtual machines imitate real systems and such must go through a booting process at first start-up. This can increase the time required for executing an application, though using check-pointing, the virtual machine state after boot up can be saved and restored once received at the grid resource[52].

Network settings within the virtual machine, if enabled, must be also automatically configured. Virtual DHCP servers or IPOP could be used in this situation as done by Grid Appliances[46].

Most applications may have bugs or cases when there execution is terminated unexpectedly. Virtual machines must be monitored during execution and report such events. Obviously the execution environment will have internal mechanisms to ensure the recovery of the application, however in some cases the entire virtual machine may need to be monitored externally.

Streaming of data needs to be setup and enabled, connections with other virtual machine units or grid resources needs to be coordinated. This could be directly controlled by the application or by a service running within the virtual machine.

3.4.4.4 Cleaning Up

Once the execution of the applications of the virtual machine is completed, and the data from the execution is forwarded onto the next virtual machine and/or data store, the virtual machine environment and virtual machine must be handled for decommissioning.

The first issue is detecting when the virtual machine has completed executing the application. This could be controlled by a service within the virtual machine, or signalled by the application. Such signals could be data transfer out completing, or the service contacting the user or workflow system. Other approaches could monitor the virtual machine performance and note the reduced CPU utilisation of the virtual machine.

Once the application has been deemed completed, the grid resource (the host machine) needs to terminate the virtual machine monitor process, hence shutting down the virtual machine. Most virtual machines will receive the shutdown signal when the virtual machine monitor process is terminated.

Finally the virtual machine images need to be handled. A simple method could presumably delete the virtual machine image from the grid resource; however a more appropriate approach may be caching the image for future use in case the user decides to rerun the workflow.

3.5 Virtual Appliances

Virtual Machine vendors began to realise the potential for using virtual machines to house the distribution of applications[63]. Known as virtual appliances, these encapsulations provide the benefits of virtual machines but also provide the developer of such applications to control the entire application stack including the operating system to be configured for the needs of their application[64].

Virtual Appliances differ from virtual machines as they provide a complete and tailored application solution within the virtual machine rather than just providing an empty virtual machine that requires an operating system installation[64].

Virtual appliances are a collection of files and a configuration file, which can be launched by the virtual machine monitor. VMWare is a strong supporter of such technologies and provides the ability to launch virtual appliances with its various virtual machine monitor software.

The advantages to developers and application distributors are that the application is running within an execution environment designed for the best performance and compatibility with the application. This is completely controlled by the vendor and user support is simplified by ensuring that the usual heterogeneous environments of hardware applications (normal applications) are avoided.

Techniques in supporting and improving virtual appliances are the use of thin-downed operating systems, known as Just Enough Operating Systems (JeOS)[65]. This thinning down is also used in FastScale's Composer Suite[66] which produces virtual machines that are tailored for the applications running within in them.

Grid Appliances is one such implementation of a virtual appliance. The virtual appliance application stack was implemented and controlled by the vendor, and for a user to make use of this virtual appliance (the grid appliance), they simply needed to execute the virtual machine on a virtual machine monitor.

The uptake and support for virtual appliances has led to the pushing of open virtualisation format (OVF) for supporting virtual appliances to be virtual machine monitor independent[67]. The OVF specifies the creation of a configuration file which details the structure of the virtual appliance with its virtual machine disk images and virtual machine hardware specifications. This file is then used by an arbitrary virtual machine monitor to then launch the virtual appliance. The OVF configuration file and accompanying virtual machine disk images are packaged into a single file (OVA).

3.6 QEMU

QEMU[37] is a type II virtual machine emulator written by Fabrice Bellard. QEMU provides full system emulation for multiple computing architectures; x86, PowerPC, ARM and SPARC; QEMU has also been ported to many different architectures such as x86, PowerPC, ARM, and SPARC [68]. QEMU provides support for running various unmodified guest operating systems.

Emulation is achieved by using a dynamic translator; however this does not provide the same performance as virtualisation. The dynamic translator converts the emulated CPU instructions into the host instruction set at runtime, and stores these translations for later usage[68]. This differs from an interpreter such that instead of interpreting every instruction it can reuse previous translations. QEMU emulates various components of a computer such as the CPU, VGA devices, serial ports, mouse and keyboards, IDE hard disks, and networking cards[68]. QEMU can also be extended to support native instruction execution virtualisation by using a device driver known as KQEMU[69].

QEMU support multiple file formats for storing virtual machine disk images. This support includes QCOW, QCOW2, VMDK and RAW. The QCOW formats provide compressed data storage for virtual machine disk images which grow as required during the execution of the virtual machine. The RAW format however is a static and the entire disk is allocated in size. QEMU provides utilities for creating and converting different virtual machine disk image formats.

The advantages of QEMU include its ability to be portably used without requiring root access to the host machine. The QEMU executable can be copied alongside the virtual machine disk image and then be launched on a system without any prior virtual machine monitor installation. QEMU also provides emulation for a wide-range of architectures which is rare in other virtual machine emulator implementations. Taking advantage of the KQEMU driver also provides the necessary performance enhancements required for making QEMU viable in running virtual machines for every day use including grid computing.

Chapter 4 Architecture of Virtual Machine Work Units

Grids are made up of heterogeneous resources that can potentially have different architectural characteristics and software configurations. Grid resources are also constrained by organisational policies defined by the grid resource owner. For an e-Scientist to successfully use the full potential of a grid they must tailor their experiment to run on all or a subset of these resources across technical and organisational boundaries. The grid application is designed for the grid rather than the grid supporting the needs of the application. In most cases an e-Scientist may have some experience in software development. However, their main concern is in their field of research. The process of developing and deploying software across a range of platforms, configurations and organisational boundaries is challenging for e-scientists (Section 2.5).

In response, a flexible grid virtual machine architecture was designed to support the dynamic generation and execution across grid resources. Utilising platform virtual machines and the advantages they provide as discussed in Section 3.4.3, Virtual Machine Work Units can be easily created to support a wide range of grid applications and their requirements.

This project aims to provide the necessary tools for e-scientists to easily and dynamically generate and execute Virtual Machine Work Units on the grid. This includes defining a flexible architecture for the Virtual Machine Work Units, providing a utility to package grid applications in to virtual machines, and a mechanism to launch the Virtual Machine Work Units on existing grid infrastructure.

4.1 Requirements for Supporting Virtual Machines in Grid Computing

Facilitating to the needs of the e-scientist was a critical concern with the design of Virtual Machine Work Units and their supporting utilities. The following requirements are essential in the design of supporting the dynamic generation and execution of Virtual Machine Work Units on the grid:

1. Allow the execution of an e-scientists grid application in a virtual machine without the constraints present on existing grid infrastructure.
2. A common virtual machine structure which allows the easy combination of grid applications and virtual machines to form a Virtual Machine Work Unit.
3. Simplistic approach for the automatic generation of Virtual Machine Work Units, allowing the e-scientist to provide their own environments or utilising pre-configured environments.
4. Supporting reuse of Virtual Machine Work Unit components and allowing the extraction of specific components with minimal effort.
5. Utilisation of the designed architecture on existing grid infrastructure with minimal or no modifications.
6. Launching and executing Virtual Machine Work Units on existing grid infrastructure and supporting the passing parameters and files to each instance.

Supporting these requirements needed the design of a flexible architecture that could be utilised on existing grid infrastructures. The use of Virtual Machine Work Units is very similar to traditional grid application execution and is discussed in detail in the next section.

4.2 Grid Architecture with Virtual Machine Work Units

Current implementations of grids make use of remote job execution systems in which jobs submitted by the user are executed across grid resources. Grid middleware and infrastructure such as Sun Grid Engine and Nimrod provide this functionality as discussed in Section 2.2.1 and Section 2.4.1 respectively. The resources are selected by the job execution system at run-time and reduce the effort required to deploy grid applications that meet the architectural specifications of the grid resource. These jobs contains a list of instructions that usually include the launching of an e-scientists application, often accompanied by job instance specific data that differs it from other cloned jobs being executed in parallel.

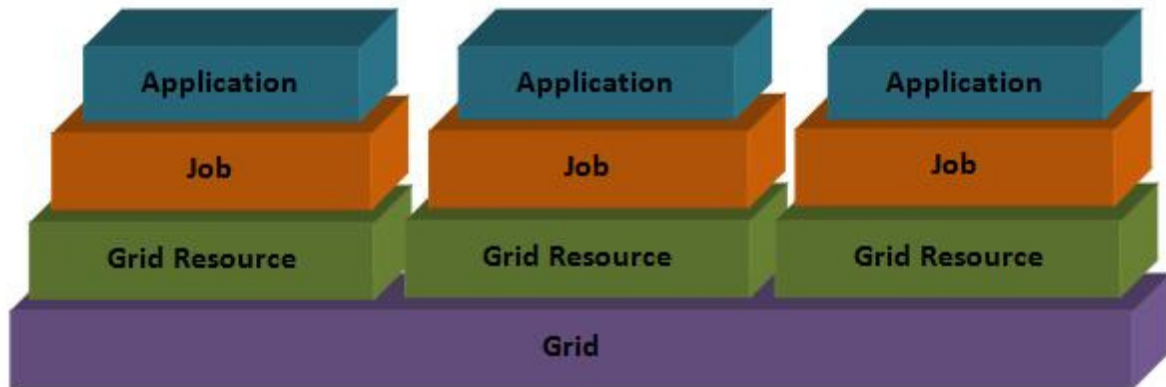


Figure 4.1: Traditional method of launching applications over the grid

The traditional model of executing grid applications on existing grid infrastructure is shown in Figure 4.1. The grid application runs directly on the grid resource and needs to meet the constraints of the grid resource.

Using virtual machines as work units applies the same architecture as traditional grids, however instead of instructions for launching the application; the jobs now contain instructions for launching the Virtual Machine Work Unit that contains the e-scientists execution environment and application; this is the same architecture as discussed in Section 3.4.3. Building upon existing grid infrastructure, Virtual Machine Work Units can be easily launched on grid resources allowing heterogeneous resources to be utilised.

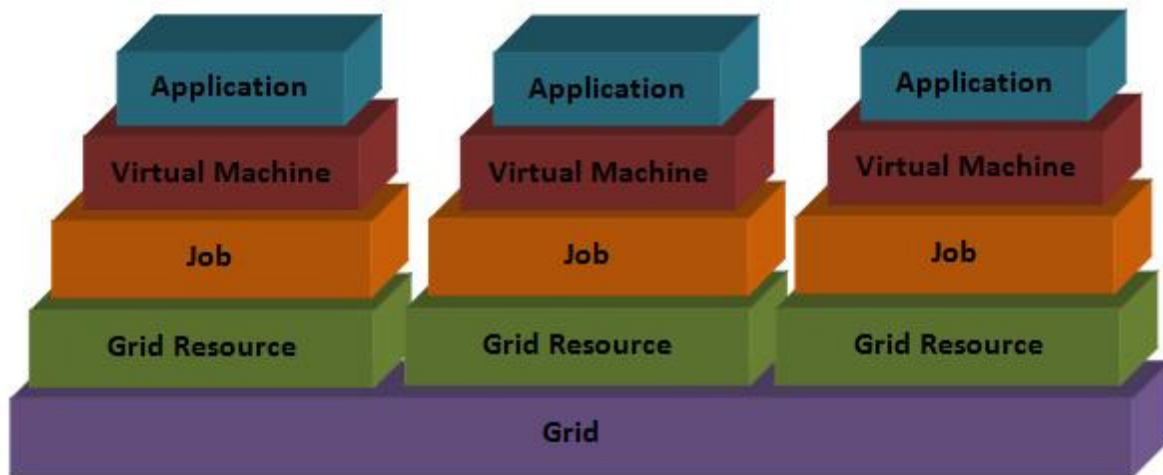


Figure 4.2: Method of launching applications through Virtual Machine Work Units over the grid

The virtual machines are assuming the position of the application on the grid as demonstrated in Figure 4.2. Therefore, the e-scientist can avoid the responsibility of ensuring the compatibility of the application (virtual machine monitor) on the grid resource and concentrate on the development the grid application.

Virtual Machine Work Units have no direct coupling with the underlying grid infrastructure as they are treated as standard jobs when the virtual machine monitor is passed along with the Virtual Machine Work Unit. Virtual machine monitors on the host can also be utilised to provide the performance gains of virtualisation rather than emulation. Such architecture of using virtual machines on grid resources has previously been utilised in other implementations that make use of Java and/or .Net where application level virtual machines are used to run applications as discussed in Section 3.4. This approach rather executes platform-level virtual machines on the grid resources and encapsulates the grid application and its data within the virtual machine.

The Virtual Machine Work Units need to be designed so that the virtual machine is independent from the virtual machine monitor so that different virtual machine monitors could be running on different grid resources. A launching mechanism is provided with the front-end to facilitate the gathering of the virtual machine monitor and to start the execution of the Virtual Machine Work Unit on the virtual machine monitor. Virtual Machine Work Units are preconfigured before being submitted, see Section 4.3. However when launching the Virtual Machine Work Units, e-scientists may require a method of passing parameters to the application running within the virtual machine at run-time. This is handled by the launching mechanism that creates a package that is accessed by the virtual machine. Once the Virtual Machine Work Unit has completed execution its output is then passed back to the e-Scientist to be used.

4.3 Virtual Machine Work Unit Structure

The structure of the virtual machine is made up of its combined virtual machine disk images and/or configuration file that details how these components interact. These virtual machine disks are then used by virtual machine to access the guest operating system, contain the e-

scientists grid application and data, and a place for outputting the results of the application execution. Designs of virtual machine executed on grid resources can differ in several ways.

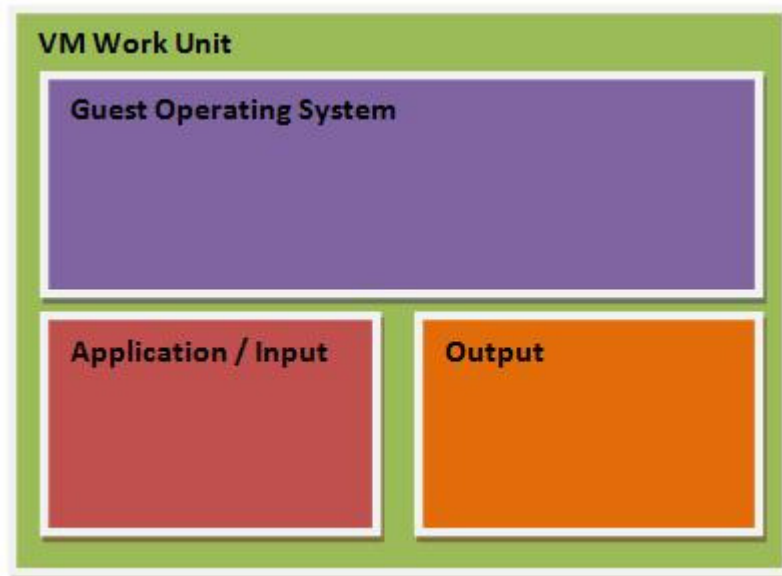


Figure 4.3: Virtual Machine Work Unit structure

Figure 4.3 shows the structure of the Virtual Machine Work Unit utilised in this project. The Virtual Machine Work Unit is made up of three virtual machine disk images: the guest operating system image, the application / input image, and an output image.

The guest operating system is represented as a separate virtual machine disk image. Referred to as a base image, the guest operating system is rarely modified from its original state and such keeping the guest operating system separate from the e-scientists application, data, and output allows the ability to reuse base images. The selection of a base image can be provided by the e-scientist or selected interactively and/or automatically based on the e-scientists grid application.

The e-scientists application and any necessary data files are encapsulated within its own virtual machine disk image. This is then combined with the base image to form an operating system with the e-scientists application. The application can be simply a stand-alone executable and/or a fully fledged application with dependencies and installation requirements. However the application needs to be compatible with the chosen base image guest operating system.

Finally the output of the application is stored within its own virtual machine disk. This separate encapsulation of data allows the results from the application to be easily transferred across the grid without requiring the overhead of the base image and accompanying application.

The Virtual Machine Work Unit is also designed to be independent of the virtual machine monitor required to launch it, providing the portability to execute over different resources with different characteristics. The virtual machine disk images are accompanied with configuration files will be aligned with the Open Virtualisation Format (OVF) standard. In an

essence a Virtual Machine Work Unit is the same as a Virtual Appliance, as instead of bundling a single application it bundles a complete system; however it differs from traditional Virtual Appliances as they are a non-interactive sequential execution of a virtual machine. A Virtual Machine Work Unit is often instantiated multiple times in parallel across a computing grid.

The Virtual Machine Work Unit is packaged the same way as an Open Virtualisation Format (OVF) virtual appliance and ensures that the Virtual Machine Work Unit can be executed in the same manner as a virtual appliance. Support for the OVF is designed into the architecture of Virtual Machine Work Units though has not been implemented at this stage.

4.4 Tailored Execution Environments within Virtual Machine Work Units

Platform virtual machines provide the functionality to emulate a complete computing environment. This computing environment can be the same architecture of the host and can run any operating system required by the e-Scientist. Another advantage of virtual machines includes the capabilities to emulate other system architectures that are different to the host's underlying system architecture at the expense of performance. If an e-Scientist application was developed for a legacy system and/or architecture other than the architectures available on the grid, the utilisation of virtual machines can allow the grid application to be executed across any grid resource regardless of architecture and operating system.

The guest operating system can be configured and controlled directly by the e-Scientist who has complete access to the machine and operating system. This includes specifying the type of operating system and particular version suited for their grid application implementation. Additional utilities and libraries can also be installed within the execution environment without the intervention of grid administrators. Grid applications that require root access can be tailored for by using virtual machines. This also provides security and containment that isolates the e-Scientists application from grid resources. Overall this reduces the issues in deploying grid applications faced by e-scientists and grid administrators, and ensures that delays in updating and installing required dependencies on grid resources are no longer present.

Modifications to the base image ensure that commands passed by the e-scientists are executed within the Virtual Machine Work Unit at execution time on the grid resource. This also provides the necessary mechanisms for ensuring that the application is loaded within the guest operating system and has access to other components of the Virtual Machine Work Unit.

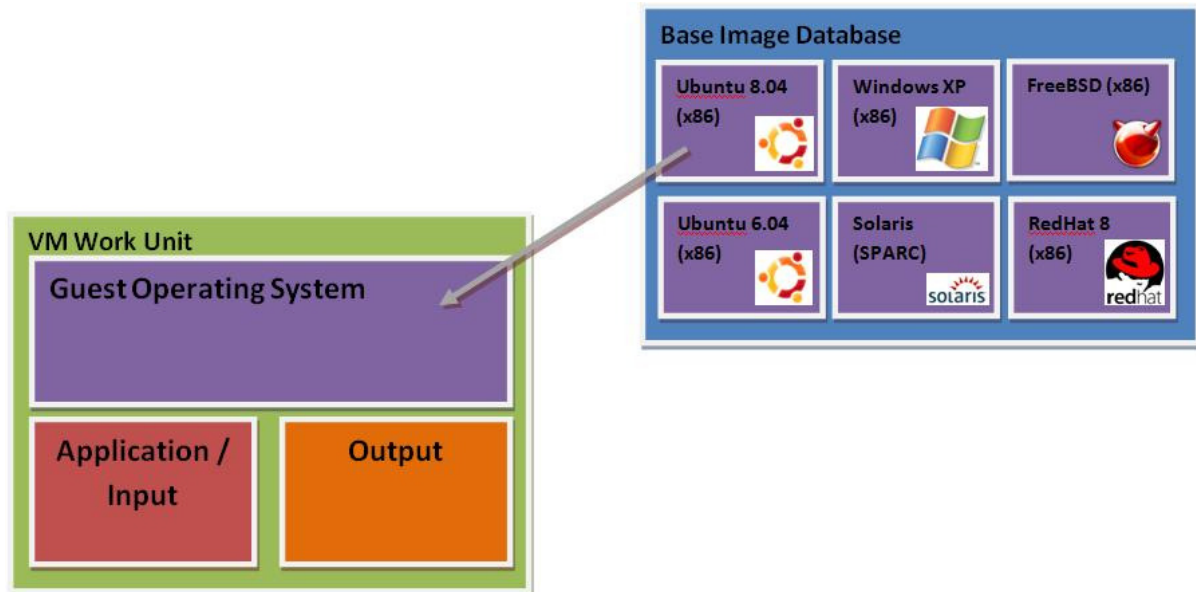


Figure 4.4: Base images do not necessarily need to be configured by e-scientists

To reduce the effort on the e-scientists behalf preconfigured base images can be chosen with execution environments already setup to automatically launch grid appliances. This can be seen in Figure 4.4, where the e-scientist can select from a range of preconfigured guest operating systems of varying architectures, operating systems, and versions.

4.5 Typical Scenario Usage of Virtual Machine Work Units

The process of required by the e-scientist needs to be as simple as possible for utilising virtual machines on existing grid infrastructure. However flexibility and control is also crucial for allowing e-scientists to have complete control when creating Virtual Machine Work Units.

The typical scenario of e-scientist using Virtual Machine Work Units follows a simple process for creating Virtual Machine Work Units and utilises existing grid infrastructures for the deployment and launching of Virtual Machine Work Units.

1. E-scientist sources the grid application they wish to use for their experiment and determine the execution environment requirements needed.
2. The settings defined are then passed to a packaging mechanism which parses this information.
3. The packager uses the information to find a suitable base image that provides the required environment by the grid application. The launching script is then generated for the guest operating system, and application specific options are configured for the environment. The application, any required data, and the launching script is encapsulated in to the application image. An output image is generated based on the size requirements of the grid application.
4. All images are then combined into a single package which is provided to the e-scientist.
5. The e-scientist then takes the Virtual Machine Work Unit and uploads it to the grid being utilised for their experimentation.

6. Using existing grid infrastructure, the e-scientist launches the Virtual Machine Work Unit using a designed launcher through the grids remote job submission system.
7. The launcher takes the Virtual Machine Work Unit and creates a ISO image to pass any parameters or files needed by the grid application specific to this job submission.
8. The Virtual Machine Work Unit is executed on a sourced virtual machine monitor. Within this environment the grid application is also executing.
9. Once the execution of the Virtual Machine Work Unit is complete, the launcher extracts the output image and cleans up any remaining files left over from the execution of the Virtual Machine Work Unit.
10. The output image is then copied back to the root node of the grid for the e-scientist to process.

Overall this architecture is designed to utilise the advantages of platform virtual machines for developing and deploying grid applications. It was essential to make the use and process of Virtual Machine Work Units as seamless and simple as possible but still provide the power for defining complex execution environments for grid applications with obscure requirements.

Chapter 5 Design and Implementation of Virtual Machine Work Units and Supporting Utilities

Virtual Machine Work Units provide the flexibility needed by e-scientists in creating and deploying grid applications, however the configuration of such environments can be challenging in its own right.

Supporting tools were designed and implemented to allow the easy creation of Virtual Machine Work Units, and the ability to easily launch Virtual Machine Work Units on existing grid infrastructure:

- Packaging utility that the e-scientist can utilise in the creation of Virtual Machine Work Units. The packager utility takes the grid application and auto sources a pre-configured base image that meets the architectural and operating system requirements, and then creates the necessary components that form the Virtual Machine Work Unit.
- Launcher utility that wraps around the Virtual Machine Work Unit, providing an interface for e-scientists to pass in parameters for each instance of the grid application launched inside the Virtual Machine Work Unit. The launcher was designed for grid infrastructure that doesn't support any inbuilt virtual machine monitors.
- To support both these utilities, a Virtual Machine Work Unit framework was developed to abstract the interactions of creating and modifying virtual machine images, and support for virtual machine monitors.

To understand why the packager and launcher utilities and the framework were developed we will look at the detailed design and implementation of Virtual Machine Work Units.

5.1 Virtual Machine Work Unit

The initial approach taken for the implementation Virtual Machine Work Units was to take all the components and place them into a single virtual machine disk that would be launched on the grid. This simple prototype would take the initial base image and copy the application into a directory within the main virtual machine disk and file-system of the guest operating system.

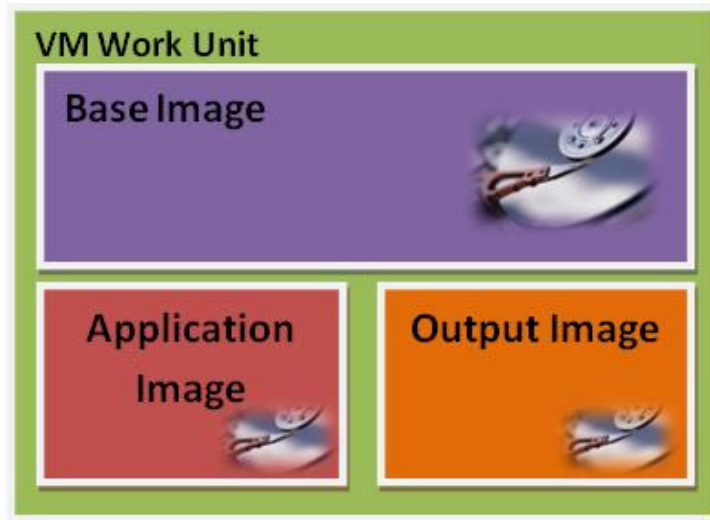


Figure 5.1: The virtual machine disk components that make up a Virtual Machine Work Unit

This simplistic approach was swapped for a more flexible and less data intensive design that allowed the guest operating system, application and input, and output to be kept separate. These components would then be combined when the Virtual Machine Work Unit is instantiated on the grid resource to create the execution environment. The components, as seen in Figure 5.1, are the base image which contains the guest operating system, the application image that contains the grid application, and the output image which is empty but provides a place for the grid application to store results.

These components once generated are packaged into a single Virtual Machine Work Unit package. The single file follows the design of the Open Virtualisation Format (OVF) virtual appliances and is the collection of virtual machine disks packaged into a single file. This file can then be compressed to reduce the file size of the Virtual Machine Work Unit. In the future, the OVF configuration file will also be contained within this package to fully align Virtual Machine Work Units with the OVF standard. At this stage the components of the Virtual Machine Work Unit are RAW virtual machine disks, however support for other formats is discussed in Section 5.4.1.

The structures of Virtual Machine Work Units were designed to be flexible, and this has allowed the easy implementation and utilisation of supporting utilities. The following sections detail the individual implementation for each virtual machine disk component.

5.1.1 Base Image

The base image component of the Virtual Machine Work Unit is the largest component of the entire package. It contains the guest operating system that meets the architectural and operating system requirements of the grid application.

The base image is the main and master virtual machine disk of the virtual machine and needs to be large enough to contain the entire guest operating system. This virtual machine disk needs to provide the necessary configurations as normal master hard-disks that contain bootable operating systems.

- Partition Table – The base image virtual machine disk needs to be partitioned and the partition type information for each partition needs to be defined.

- Master Boot Record – The MBR on the main virtual machine disk is required to direct the BIOS of the virtual machine to the boot loader to launch the guest operating system.
- Boot Loader – The boot loader is responsible for launching the guest operating system. It needs to be setup to allow the automatic loading of the desired operating system. Minimising this time can reduce the static time in launching Virtual Machine Work Units.
- If guest operating system is a Unix-based system, a partition needs to be allocated for swap space. This is usually optimised to the same size of the memory provided by the virtual machine however this is not always the case. The nature of varying requirements means that the memory size is dynamic and can differ depending on the grid application and/or the grid resource.
- All partitions need suitable file system initialisation unless partitions specifically are used in raw such as swap partitions. This file system is usually prescribed by the guest operating system or specifically configured to another format by the e-scientist in pre-configured environments.

Fortunately most of the above hard-disk configurations are automatically configured with the initial installation of the guest operating system within the virtual machine. However these need to be guaranteed before utilising such environments for use as Virtual Machine Work Units.

The guest operating system within the virtual machine also needs to be configured to support other mechanisms required to automatically launch grid applications and combining various Virtual Machine Work Unit components. This is discussed later in Section 5.1.4.

5.1.2 Application Image

The application image is small sized virtual machine disk that contains application specific files and data. The application image also contains the script-bin which contains any scripts that are to be launched at the start up of the virtual machine.

The size of the application image is dependent on a number of items:

- The size of the application, and its supporting files.
- Any static data provided to the grid application when being packaged.
- Buffer space for any temporary files or un-compressed files created by the application during execution. This buffer space is defined by the e-scientist.

This application image like the base image requires that partitions are correctly initialised. In this implementation the application image is a single partition. The file system within this virtual machine disk is formatted to meet the requirements of the base image and guest operating system. In most cases this is limited to either ext2 file system or FAT32 file system which are widely used and supported by most guest operating systems. This can be a limiting factor in the portability and reuse of application images.

There are a number of advantages of keeping the grid application in a separate image.

- If changes or updates are made to the application it can easily be copied directly in to application image or the entire application image can be substituted.

- Static data provided on a per-experiment basis can be easily accessed and substituted in to the application image.
- Separates the application from the guest operating system and allows reuse of both the application image and base image (guest operating system).

The dependencies of the application have to be installed within the guest operating system; however the installation files for these dependencies could be contained within the application image. The automatic pre-configuration of these dependencies can be automated by inserting installation instructions in to the auto-run script or by modifying the master Virtual Machine Work Unit before pushing it to be instantiated across grid resources.

Any temporary files created outside this application image and within the application image will be lost after the instantiation unless the Virtual Machine Work Unit is executed with overwriting preferences. In most cases this is desirable as the Virtual Machine Work Unit can be reused without any issues of residual files and possible corruptions that are sustained from previous executions.

5.1.3 Output Image

The output image is also a virtual machine disk, however unlike the base image and application image, the file system within the virtual machine disk is empty. The output image is implemented to provide the grid application with a place to output results and any other supporting files that are generated during the execution of the Virtual Machine Work Unit.

This output image like the base image requires that partitions are correctly initialised. In this implementation the output image is a single partition. The file system within this virtual machine disk is formatted to meet the requirements of the base image and guest operating system. In most cases this is limited to either ext2 file system or FAT32 file system which is widely used and supported by most guest operating systems.

The size of the output image needs to be predefined by the e-scientist. However output images of different sizes like application images can be substituted in to existing packages if required.

Once the Virtual Machine Work Unit has completed execution, the output image is returned to the e-scientist to extract the results. A simple supporting utility was designed that utilised the Virtual Machine Work Unit framework described in Section 5.4 which mounts and extracts the contents of the image to an output directory using a simple command-line interface.

5.1.4 Execution Environment

The above virtual machine disks, discussed in the previous sections, of the Virtual Machine Work Unit are combined to create the entire execution environment. The application image strictly holds the files and folders used by the grid application; however in the case when dependencies within the guest operating system are needed these are contained within the base image. The base image is unique to the Virtual Machine Work Unit and has been tailored and modified for supporting the grid application.

The advantage of using platform virtual machines is that it allows the e-scientist to completely control and tailor their execution environment. The e-scientist simply needs to ensure that all Virtual Machine Work Unit components are available at execution time and properly configured within the guest operating system, ensure at start up their grid application will execute, and finally once completed will automatically shutdown.

However to support reusable base images it is necessary that execution environments are setup in such a way to allow automation and flexibility for arbitrary grid applications. This includes providing a guest operating system that loads at the start-up of the virtual machine, supports multiple processes, automatic login or background services, capable of mounting or accessing hard-disks, capable of accessing ISO9660 CD-ROM, and the ability to restart or shutdown the machine.

The guest operating system needs to be pointed to by the boot loader and ensured that it is executed at the start up of the execution of the virtual machine. This is rarely an issue, however if the base image is configured with multiple guest operating systems and/or the guest operating system has different modes this will prevent the automation of Virtual Machine Work Units if not setup correctly.

Once the guest operating system starts it needs to support the execution of multiple processes. This allows the execution of multiple scripts or executables and supports grid applications that may be broken down into smaller processes and tasks.

The execution environment also needs to provide the automatic login to root access if the guest operating system doesn't support background daemons or services, and if the grid application utilises a graphical user interface. Utilising background daemons and services is required to provide the script launching service that launches scripts required to execute the grid application. A Virtual Machine Work Unit daemon was designed and implemented to support this as well as providing as initialisations required in the execution environment. This is discussed in detail in Section 5.1.4.1.

The execution environment needs to support the mounting and accessing of other virtual machine disks provided to the virtual machine. In the case where the guest operating system doesn't support the automatic mounting of these virtual machine disks and their partitions, then it must provide mechanisms to allow this. This process is handled by the Virtual Machine Work Unit daemon and by the script created when the grid application is packaged within the Virtual Machine Work Unit. More information on the virtual machine auto-run script contents is discussed in Section 5.2.5.

Similar to the above requirement but more specific, the execution environment must support the accessing of CD-ROM's in the ISO9660 format. This format limitation will be expanded if required, however most operating systems support this standard. The access to the CD-ROM allows the passing of parameters and files at the instantiation of the Virtual Machine Work Unit.

Finally the execution environment must support the shutdown or restarting of the virtual machine. This is necessary as it used to signal the completion of the grid applications execution and in most cases allows the virtual machine monitor to terminate. This then allows the launching utility to then retrieve the output image. The virtual machine auto-run script is responsible for signalling the shutdown or restart within the virtual machine.

5.1.4.1 Virtual Machine Work Unit Daemon

To support the automatic launching of grid applications a Virtual Machine Work Unit daemon was implemented. The Virtual Machine Work Unit daemon was designed to run as a background service within the guest operating system installed on the base image.

When the Virtual Machine Work Unit is started the daemon is launched within the guest operating system. This has root access and also gives any scripts launched the same permissions. This is acceptable because the Virtual Machine Work Unit is a self-contained environment and is ideally isolated from the host operating system (grid node resource). It should be noted this privilege level means that any scripts could accidentally or maliciously modify the instance of the base image; however the base image is copy and unique to that instance of the Virtual Machine Work Unit and any changes will be restored at the completion of the Virtual Machine Work Unit execution.

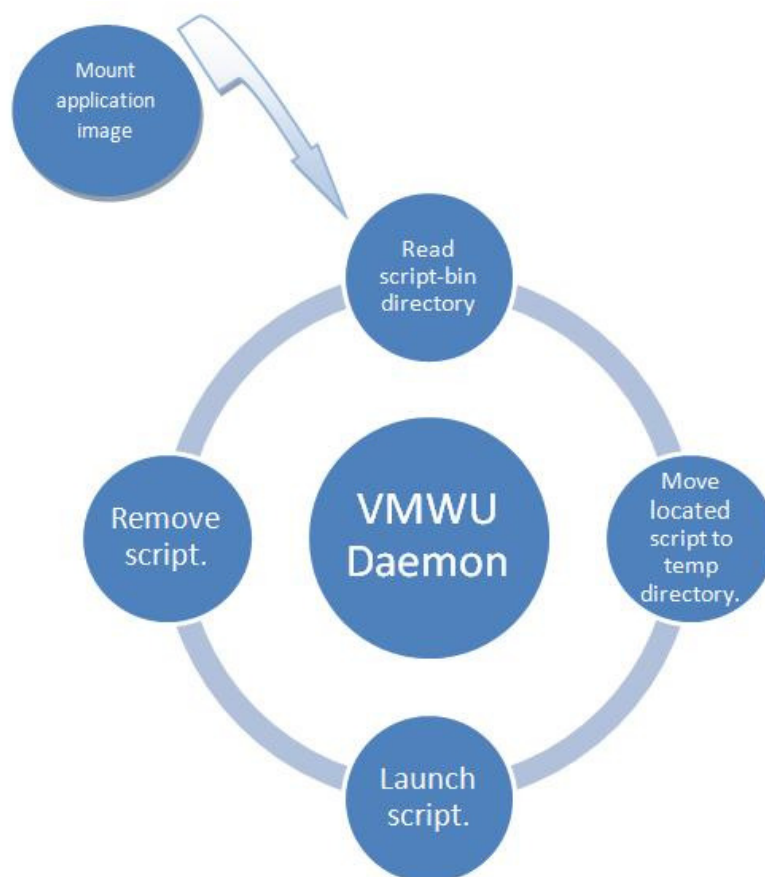


Figure 5.2: The algorithm of the Virtual Machine Work Unit daemon

The execution of the virtual machine work daemon is shown in Figure 5.2. Once the Virtual Machine Work Unit daemon is launched it mounts the application virtual machine disk

image to the guest operating system file system. This application image contains the grid application as well as a script-bin directory where waiting scripts are contained. The daemon then repeatedly polls this script-bin directory for any executable scripts.

A found script is then moved to a temporary directory and is then executed. Once it has finished executing it is removed. This process continues for each script found within the script-bin directory.

To allow flexibility only the application image is mounted by the daemon, the other components of the Virtual Machine Work Unit are mounted and loaded by the instructions contained in the main virtual machine auto-run script.

The Virtual Machine Work Unit daemon was implemented in C and designed for UNIX based systems. The executable needs to be compiled for the specific guest operating system using C compiler. A corresponding port to a Windows service is yet to be implemented.

The Virtual Machine Work Unit daemon loads its configuration settings from a file located in the same directory as the daemon's location. This can be configured to allow the administrator of the guest operating system to decide the mounting directory for the application image, the device that refers to the application image virtual machine disk, the script-bin directory location, and the temporary directory for holding executed scripts.

The Virtual Machine Work Unit daemon also uses Syslog[70] libraries that provide a logging mechanism that can be used to debug execution environments. This includes logging the start up of the daemon, initialisation, application image virtual machine disk mounting, and discovery and launching of scripts found within the script bin.

There are a number of ways of installing the Virtual Machine Work Unit daemon into the guest operating systems. The most flexible and supported method is to use the specific daemon launching mechanism provided by the guest operating system. In Ubuntu this is using init.d scripts, and in FreeBSD this is using rc.d scripts. Both these mechanisms were utilised in creating pre-configured execution environments in the testing of Virtual Machine Work Units.

5.2 Creating Virtual Machine Work Units

Due to the complexities that are present in the creation of execution environments and the need to simplify the creation of Virtual Machine Work Units led to the design of a simple packaging system that could take the e-scientists application and place it in a suitable execution environment. The resulting output of the packager would be a virtual machine monitor independent Virtual Machine Work Unit.

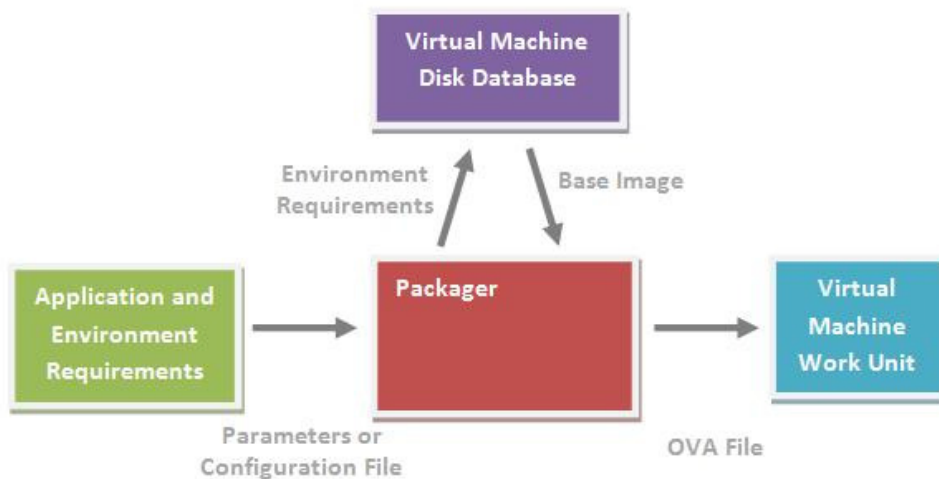


Figure 5.3: Overview of the packaging utility that allows the creation of Virtual Machine Work Units for e-scientists

The packager utility revolves around the application and configuration settings provided by the e-scientist and access to a virtual machine disk database that houses some pre-configured base images providing different architectures and operating systems. This allows e-scientists to avoid having to configure their own execution environments. The design of the packager utility can be seen in Figure 5.3.

5.2.1 Process

Constructing Virtual Machine Work Units needs to be dynamic and designed to suit the grid application being packaged. The steps involved in generating Virtual Machine Work Units are distinct and each step concentrates on specific aspects of the Virtual Machine Work Unit. This allows the inclusion of highly configurable options in each of these steps and allows the e-scientist to specify these options



Figure 5.4: The process of creating Virtual Machine Work Units

Based on the design of virtual appliances, Virtual Machine Work Units are generated by a packaging utility process is shown in Figure 5.4. The basic steps include the loading of the application and its environment requirements, the selection and loading of the base image containing the guest operating system, the creation of the virtual machine auto-run script, the creation of the application and output images, the support for Open Virtualisation Format (OVF) standards for virtual appliances, and finally the combination of all generated elements.

The packager utility initialises its environment and provides temporary directories for the contents of the application image, and the generated virtual machine disks that make up the Virtual Machine Work Unit. The individual steps will now be discussed in more detail.

5.2.1.1 Loading the Application

The first step is to load the application provided by the e-scientist. The information on the application, its location and the required execution environment configuration settings are all provided by the e-scientist. This allows the e-scientist to provide information for launching the grid application within the virtual machine, the supporting architecture, and the supporting guest operating system required for execution. The e-scientist can use their own execution environment by providing the base image through configuration settings or utilise the attached virtual machine disk database to find a base image that meets the requirements of their grid application.

The settings provided by the e-scientist can either be provided by command-line arguments and/or a configuration file. These can be used in conjunction where the command-line arguments overwrite any configuration file settings. The packaging utility also provides defaults which can be used to simplify the packaging process. These settings are parsed and the packager is set up as defined by the e-scientist.

The file and directory locations of the grid application as well as supporting data are copied to a temporary directory that represents the application image. The e-scientist can define the destination of these files within the application image root by manually providing each files source and destination or if they wish can simply use the default settings and only specify the application directory.

Once all these files are copied to the temporary directory, the application loading is complete and the sourcing of the base image occurs.

5.2.1.2 Loading the Base Image

The base image represents the guest operating system and architecture utilised by the grid application and is important to ensuring the functionality of the Virtual Machine Work Units. Flexibility provided by the packager utility gives the e-scientist two options for the loading of the base image.

For grid applications with obscure requirements and that require complex configurations the e-scientist can provide their own pre-configured base images that are suited for their grid application. This allows the e-scientist to use the features of the packager utility to load their application within the Virtual Machine Work Unit and promotes the reuse of

personalised execution environments instead of crafting a single guest operating system designed solely to execute the grid application.

For e-scientists who wish to utilise the benefits of Virtual Machine Work Units without the strain and headache of creating execution environments they can utilise existing pre-configured base images that are managed in a central location in a virtual machine disk database. The e-scientist simply needs to provide the basic architectural requirements of their grid application including its machine architecture and operating system architecture. The packaging utility then automatically finds a set of base images that meet these requirements. The e-scientist can then choose one of these base images, or allow the packaging utility to choose the best match.

It should be noted that the second option of using the virtual machine disk database does not limit the configuration of the execution environment. Once the Virtual Machine Work Unit is packaged the e-scientist can load the master Virtual Machine Work Unit and modify it as required for supporting their grid application.

Once a base image is selected, the information on this base image is then passed to the required components of the packaging utility. This includes information on the location of the base image on the host's file system, the guest operating system file system format, the location of the corresponding script template that can be used to generate the virtual machine auto-run script, and the devices used for accessing the application and output image virtual machine disks from within the virtual machine.

The base image is then copied to a temporary directory to be used later by the packager utility. Once this is completed the base image loading is complete and then the virtual machine auto-run script is generated.

5.2.1.3 Creating the Virtual Machine Auto-Run Script

The virtual machine auto-run script is the script that is launched within the virtual machine, and is required to initially setup the Virtual Machine Work Unit components (output image and the generated ISO image), launch the grid application, and shutdown the virtual machine.

The script template is a shell script; however support for other scripting languages is dependent on the guest operating system. The shell script needs to be supported by the shell that is used within guest operating system.

Information provided by the base image points to a suitable virtual machine auto-run script template that can be used to provide the automation required to launch the grid application within the virtual machine. If required the e-scientist can provide their own script template which allows the creation of complex scripting if required by the e-scientist.

Tokens within the script template are then located and substituted with Virtual Machine Work Unit specific attributes. These include the launching command, the directories that contain the application image, output image, and CD-ROM files and directories. More information on the tokens available is discussed in Section 5.2.5.

Once the virtual machine auto-run script is generated it is copied to the script-bin directory contained within the application image. This ensures that the script is automatically launched by the Virtual Machine Work Unit daemon contained within the guest operating system.

The virtual machine auto-run script is the last generated entity that makes up the application image, which now allows the application and the script to be copied into a generated virtual machine disk.

5.2.1.4 Creating the Application Image

The contents of the application image are managed in a single temporary directory created by the packaging utility. This directory contains all the files and directories of the application, any data files, and the virtual machine auto-run script generated in the last step.

The virtual machine disk needs to be dynamically created and requires the total size of the files and directories needed for the application, data, and script. Space for the execution of the application is also specified by the e-scientist to ensure that the application has enough room if it creates any temporary files and directories. All these components equate to the total size of the virtual machine disk image; however it also necessary to take into account the space required for the partition table information contained at the start of the virtual machine disk.

The packager uses the virtual machine work framework virtual machine disk abstractions as discussed in Section 5.4.1. This abstracts the technical implementations of creating the virtual machine disk. The packager uses the virtual machine disk format that corresponds to the base image. For example if the base image is in RAW virtual machine disk format, the generated application virtual machine disk is in the same format. The file system format also corresponds to the file system format specified by the base image either provided by the e-scientists or as part of the virtual machine disk database.

Once the virtual machine disk is created, the packaging utility then proceeds to copy the files from the temporary application image directory to the virtual machine disk. This virtual machine disk is mounted to the host operating system file system and uses the file copying mechanisms provided by the operating system to transfer files.

The virtual machine disk is unmounted at the completion of the file transfers and completes the creation of the application image. The location for the grid application to output is then created.

5.2.1.5 Creating the Output Image

The grid application requires a location to output its results to, and a place where other supporting files such as standard out and standard error streams can be redirected to. The results also need to be easily extracted from the Virtual Machine Work Unit without the overhead of going through the base and application images. As such a separate virtual machine disk is created to store the generated files during the execution of the Virtual

Machine Work Unit. This virtual machine disk needs to be large enough to support any output, however this size is not exact and needs to be estimated by the e-scientist.

The packaging utility takes this estimation by the e-scientist and uses the Virtual Machine Work Unit framework virtual machine disk to create the output image. This matches the virtual machine disk format of the base image. The file system format also corresponds to the file system format specified by the base image either provided by the e-scientists or as part of the virtual machine disk database.

Due to the nature of the output image, the virtual machine disk does not need to be mounted for file transfer as the output image is normally an empty virtual machine disk. As all virtual machine disks have been created and defined, the configuration file of the Virtual Machine Work Unit can be created.

5.2.1.6 Creating the Virtual Machine Work Unit Configuration File

The final stage before the Virtual Machine Work Unit components are combined is the creation of a configuration file that specifies the arrangement and specifications of the virtual machine that represents the Virtual Machine Work Unit.

Following the design of virtual appliances the configuration file would allow the e-scientist to define the virtual machine specifications such as the required memory, and description of the supported devices within the virtual machine. Following the standards defined by the Open Virtualisation Format (OVF) as discussed in Section 3.5, the Virtual Machine Work Unit can be launched on OVF compliant virtual machine monitors. This allows the virtual machine monitor to take the separate virtual machine disks and configures a virtual machine that meets the specification and makeup of the Virtual Machine Work Unit as required by the e-scientist. This removes a lot of the effort in providing the portability necessary for Virtual Machine Work Units.

However with the current implementation, the support for the OVF standard is not present and no configuration files are generated. This does not limit the portability but requires the accompanying launching utility to organise the Virtual Machine Work Unit components together for the virtual machine monitor. The e-scientist is also required to specify the architecture of the virtual machine to the launcher.

Now all the components of the Virtual Machine Work Unit have been generated and need to be packaged in to a single package.

5.2.1.7 Packaging Virtual Machine Work Units

The Virtual Machine Work Unit components (base image, application image, output image, and configuration file) once generated were placed in a single temporary directory. The package that contains the virtual machine disk is simply a TAR file of all these components together. The Virtual Machine Work Unit may or may not be compressed at this level depending on the requirements of the e-scientist.

To ensure compatibility with OVF standards, the generated TAR file is exactly the same as an Open Virtual Appliance (OVA) file which states that the configuration file must be the first

component combined followed by the virtual machine disks. This allows virtual machine monitors and other utilities to read information about the virtual appliance without extracting all components.

Once packaged the Virtual Machine Work Unit is complete and ready to be executed on the grid. However the e-scientist may wish to further define the execution environment or test the Virtual Machine Work Unit.

Given this process, the design of packager utility is presented in the following sections.

5.2.2 Design of the Virtual Machine Work Unit Packaging Utility

The design of the packaging utility uses an object-oriented approach to generating the Virtual Machine Work Units. The packaging utility was also designed to responsible for most of the work of generating the Virtual Machine Work Unit, only requiring the e-scientist to pass their grid application and requirements, with the packager and pre-configured base images handling the rest.

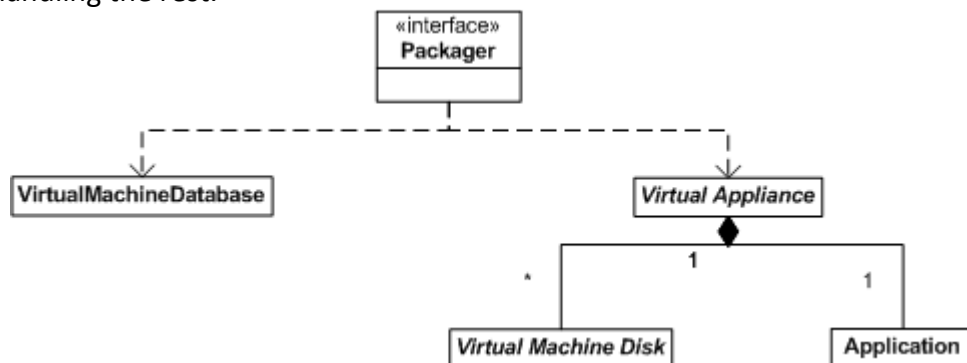


Figure 5.5: Design of the Virtual Machine Work Unit packager utility

The overall design of the packaging utility is shown in Figure 5.5. The packaging utility interfaces with a virtual machine database class that allows the packager to access the virtual machine disk database and source pre-configured base images. More information on the virtual machine disk database can be found in Section 5.2.3. The packaging utility also interfaces with a virtual appliance class which contains the methods required in generating a Virtual Machine Work Unit. The virtual appliance is made up of two other classes, the application class that represents the grid application being packaged and the virtual machine disks that make up the components of the Virtual Machine Work Unit.

To support portability and flexibility of Virtual Machine Work Units, the packager uses a virtualisation framework that was implemented to abstract virtual machine disks and virtual machine monitors. More information on the framework is discussed in Section 5.4.

The packager is implemented in Perl for Linux-based systems and makes use of several Perl Modules and GNU command-line utilities. A full listing of dependencies can be found in Appendix C. Future versions of this utility will be developed to support the creation of Virtual Machine Work Units under windows. At this stage the utility requires root access for the creation and modification of virtual machine disks and cannot be supported and installed on the root node of the grid without root access for mounting devices. This requires the packaging utility to be installed and used on the e-scientists workstation which

provides some advantages such as testing of the execution environment and Virtual Machine Work Unit.

5.2.2.1 Virtual Appliance

The virtual appliance class was designed to support the corresponding steps outlined in the process of creating Virtual Machine Work Units as discussed in Section 5.2.1.

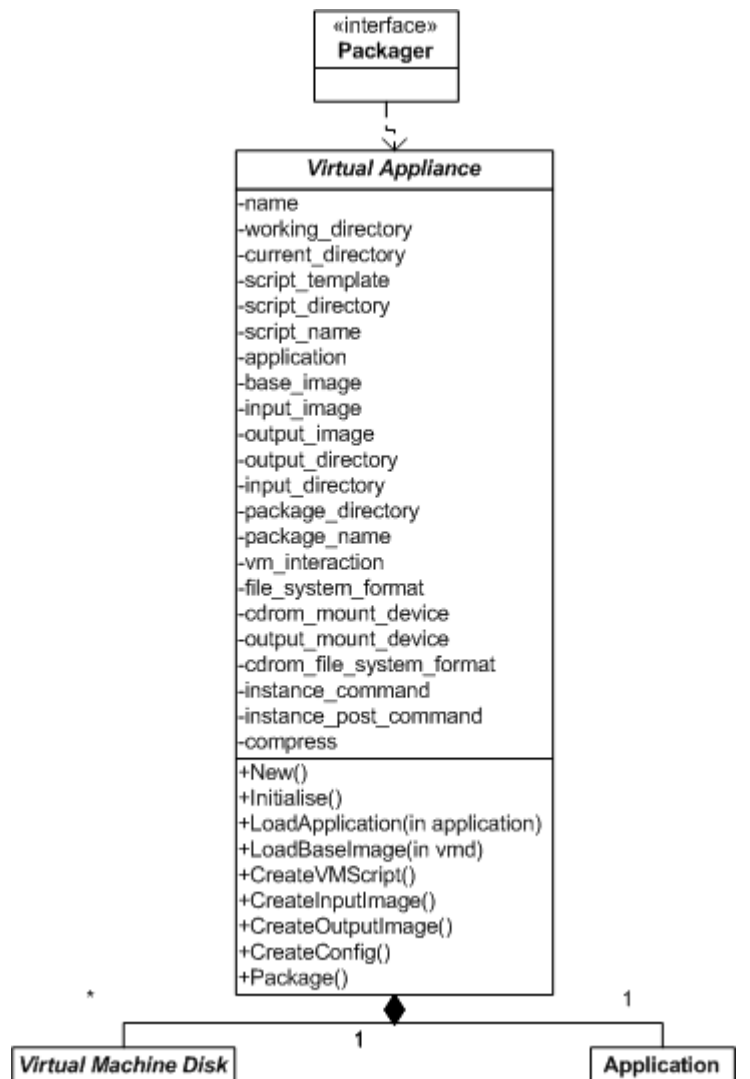


Figure 5.6: Design of the virtual appliance class used by the packaging utility

The structure of the virtual appliance class can be seen in Figure 5.6. The attributes of the virtual appliance include references to the virtual machine disks that make up the components of the Virtual Machine Work Unit. It also contains a reference to the application class. Other attributes are used for the creation of the virtual machine auto-run script.

The methods of the virtual appliance class directly relate to the steps required to construct the Virtual Machine Work Units. These methods need to be called in the correct order. At this stage support for the creation of the configuration file is not present.

5.2.2.2 Application

The application class was designed to represent the grid application provided by the e-scientist and contains information on the application.

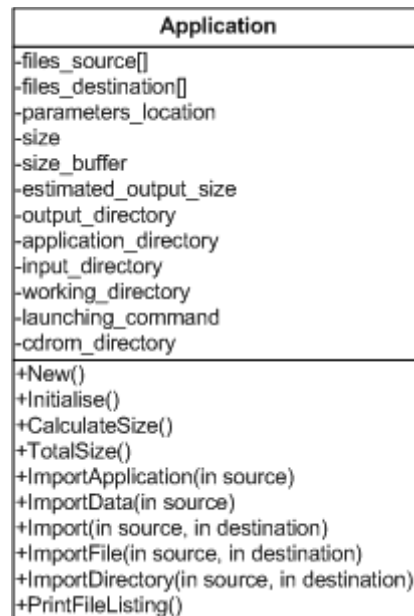


Figure 5.7: Design of the application class used by the packaging utility

The design of the application class can be seen in Figure 5.7. The application image contains two arrays that detail the files to be copied to the application image of the Virtual Machine Work Unit and their corresponding destination within this application image.

Supporting methods are provided to allow the importing of files and directories, as well as methods to calculate the total size of the application image.

5.2.3 Virtual Machine Disk Database

Providing pre-configured base images was critical to ensuring the easy creation of Virtual Machine Work Units. To allow the automated sourcing of such base images a virtual machine disk database was implemented.

The database utilised by the current packager utility is limited to a SQLite3 flat file database and/or Perl DBI supported databases. The database is accessed through a virtual machine disk database object and such can be used to hide the database implementation specific details.

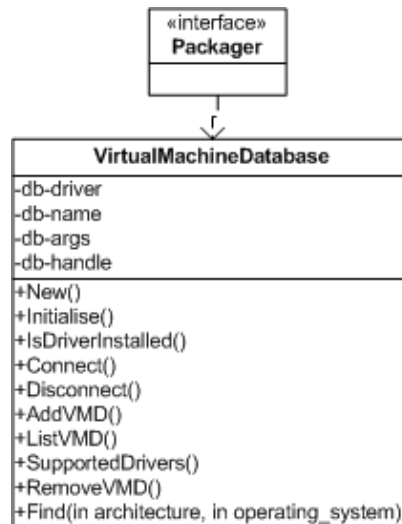


Figure 5.8: Design of the virtual machine disk database class used by packaging utility

The class, as shown in Figure 5.8, provides an interface to the database. It allows the adding and removal of base images from the database, the print out of a listing of all base images contained within the database, supported drivers for using the DBI interface, and the ability to find base images within the database that meet the architecture and operating system specifications required by the grid application.

Table 5.1: Schema of the virtual machine disk database

Database Field	Description
VMD_ID	A unique ID that identifies the base image.
VMD_NAME	The name of the base image, normally named after the guest operating system and version number. E.g. Ubuntu 8.04
VMD_DESCRIPTION	Provides a description on the base image execution environment and also includes login details and passwords.
VMD_LOCATION	Specifies the file location of the base image.
VMD_ARCHITECTURE	Specifies the architecture the base image was created for. E.g. x86
VMD_GUEST_OS	Specifies the architecture of the guest operating system. E.g. linux2.6
VMD_FILESYSTEM	Specifies the file system format the base image supports.
VMD_INPUT_DEVICE	Specifies which device points to the application image virtual machine disk.
VMD_OUTPUT_DEVICE	Specifies which device points to the output image virtual machine disk.
VMD_CDROM_DEVICE	Specifies which device points to the CD-ROM of the virtual machine.
VMD_SCRIPT_TEMPLATE	Specifies the file location of the script template designed for the base image and guest operating system.
VMD_DATE_ADDED	Provides the date of when the virtual machine was added to the virtual machine disk database.

The database contains information on the base image including its configuration, and any supporting files needed by the packaging utility. Table 5.1 shows the schema of the virtual machine disk database and a description of the information it contains.

5.2.4 Interface and Configurable Options

The packager was designed to be as flexible as possible providing the power to generate complex environments if required. However it was also necessary to make it simple for e-

scientists to create and utilise simple environments for their grid applications. The e-scientist can interact with the packaging utility through a configuration file and/or parameters passed through the command-line. This can be used in conjunction with parameters overwriting any settings pushed in by the configuration file provided by the e-scientist.

The format of the configuration file is simply the “option=value” and the corresponding command-line argument is “-option <value>” where value is optional depending on the option being set.

The most simple and common configuration options that are required for generating a Virtual Machine Work Unit are:

- *package-name* – Specifies the name of the Virtual Machine Work Unit outputted by the packager.
- *app-import-dir* – Specifies the directory of the files and sub-directories that contain the grid application. Any contents within this directory will be transferred to the application image of the Virtual Machine Work Unit. The directory specified will be the root level of the application image.
- *app-launch* - Specifies the launching command for the grid application. If the grid application output is directed to standard out then this launching command should include standard out redirection that points to the output directory specified by “app-output-dir”. Redirecting standard error may also be useful in debugging any grid application errors that occur during the instantiation of the Virtual Machine Work Unit. If the application requires parameters at run-time than this can be accomplished by appending “`cat /cdrom/param`” to the launch command. The application executable should be the absolute path, e.g. /bin/sleep. Represented as a string.
- *app-architecture* - Specifies the required architecture of the grid application. Options are dependent on the virtual machine disk database; however in the test implementation an example includes “x86” architecture. This value is checked against the database.
- *app-os* - Specifies the required operating system of the grid application. Options are dependent on the virtual machine disk database; however in the test implementation an example includes “linux2.6” based operating system. This value is checked against the database.
- *base-image-autosource* - Specifies if a pre-configured base image should be sourced to meet the requirements of the grid application. This removes the effort required by the e-scientist in creating an execution environment.
- *base-image-autosource-interaction* - Specifies if when sourcing a pre-configured base image, the e-scientist should choose from a list of possible environments. If set to false, the first possible match is selected. Used in conjunction with the “base-image-autosource” option.
- *script-template* - Specifies the location of the template file used to create the virtual machine auto-run script. This is usually dependent on the base image used. If the base image is auto sourced then this information is retrieved from the virtual machine disk database. Using this option will overwrite the template script used.

- *edit-script* - Specifies if the packager should launch a text editor to modify the virtual machine auto-run script before packaging the Virtual Machine Work Unit. Allows the e-scientist to provide any more commands that are needed by the grid application.
- *vmddb* - Specifies the location or hostname of the virtual machine disk database.
- *input-import-dir* - Specifies the directory that contains any data that needs to be bundled with the Virtual Machine Work Unit. Any files or sub-directories within this directory are copied to the application image and contained in a directory called "data".
- *edit-vm* - Specifies if the e-scientist wants to edit the execution environment before the Virtual Machine Work Unit is packaged.
- *compress* - Specifies if the Virtual Machine Work Unit generated should be compressed.

Appendix A details the complete options of the packaging utility available to the e-scientist and a description detailing the behaviour of each of the options.

5.2.5 Virtual Machine Auto-Run Script

The virtual machine auto-run scripts generated by the packaging utility provide the necessary interface to define the behaviour and execution of the grid application within the Virtual Machine Work Unit when being executed across the grid.

The script template provides the basic constructs required for providing access to each of the components of the Virtual Machine Work Unit, launching the grid application, and ensuring the shutdown of the virtual machine.

Script templates are highly coupled with the guest operating system contained in the base image and needs to be able to be launched within this guest operating system. It is important and required that when base images are preconfigured a script template is provided to be utilised by the packaging utility. This limits the wide reuse of script templates, however some operating systems share common shell scripting environments and in some cases these script templates can be reused.

To provide the flexibility of reusing virtual machine auto-run script template, tokens are placed within the script template and are substituted by the packaging utility to support the requirements of grid application within the Virtual Machine Work Unit.

At this stage a number of tokens can be placed in to script templates. This allows the e-scientist to tailor the script to meet the needs of their grid application. The available substitutions are available:

- Application directory
- Input directory
- Output directory
- Output file system format
- Output mount device
- Launching command
- Working directory
- CD-ROM directory

- CD-ROM mount device
- CD-ROM file system format
- Input mount device
- Instance command
- Instance post command

A full description of the token substitutions available is listed in Appendix B.

The constructs required for generating Virtual Machine Work Units has been discussed. Support for launching Virtual Machine Work Units is also required for supporting Virtual Machine Work Units on existing grid infrastructure. The launching utility will now be discussed in detail.

5.3 Launching Virtual Machine Work Units

Once the Virtual Machine Work Unit has been generated the e-scientists now has the ability to launch this across multiple grid resources using existing grid infrastructure. However the nature of grid applications usually means that each grid application instance on the grid has different execution behaviours depending on its input. It was imperative that support for passing parameters and files to a run-time instance of a Virtual Machine Work Unit was available for the e-scientist to use. This promoted the reuse of Virtual Machine Work Units instead of being a disposable wrapper for the grid application.

The virtual machine monitor used to execute the Virtual Machine Work Unit also poses an issue when the grid resource does not have virtual machine monitor installed locally within the grid resource.

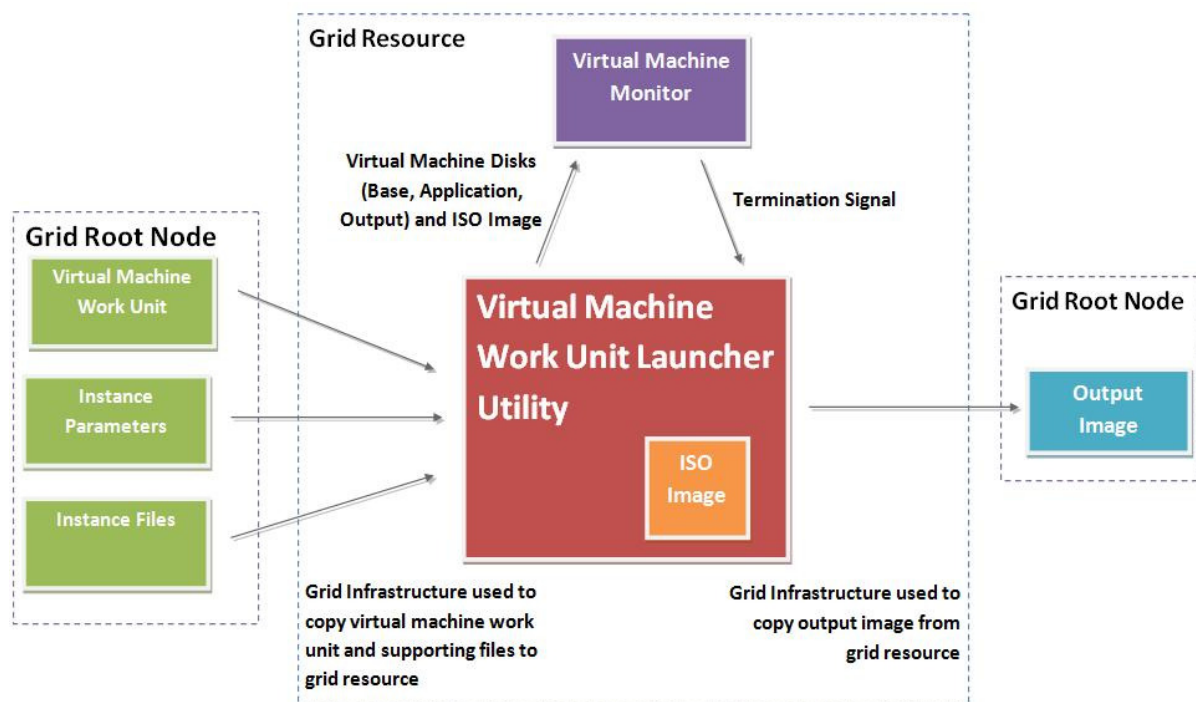


Figure 5.9: Design of the Virtual Machine Work Unit launcher utility

As such, a Virtual Machine Work Unit launching utility was developed to provide the interface to the abstracted virtual machine monitor and the ability for the e-scientist to easily pass parameters and any other files required by the instance of the Virtual Machine Work Unit.

The overall design of the Virtual Machine Work Unit launcher can be seen in Figure 5.9. The launcher takes the Virtual Machine Work Unit, parameters, and any files that are to be passed to the grid application. The launcher then passes the parameters and files provided by the e-scientist to the Virtual Machine Work Unit. The Virtual Machine Work Unit is then launched on the virtual machine monitor, and once the virtual machine monitor terminates the launcher extracts the output image.

The launcher is run within the grid resource, and the mechanisms used to transfer the Virtual Machine Work Unit, instance files, and the returning of the output image are controlled by the grid infrastructure.

The Virtual Machine Work Unit launcher needs to be available at each grid resource. The simplest approach to providing access to the launcher is for the e-scientist to keep the launcher utility within a directory in the home directory. Most grid infrastructures provide access to a global user directory. If required the launcher utility can be installed on each of the individual resources, however this is not necessary if the previous requirement is made available.

The virtual machine monitor used in the current implementation is QEMU, which emulates the virtual machine. The advantage of using QEMU is that it can easily be used on a grid resource by simply copying the QEMU executables to the grid resource. This was made easier when the same method of utilising the global home directory was used to contain QEMU. There are some issues with this approach such as ensuring that QEMU was compiled for the grid resources, and providing virtualisation drivers. QEMU emulates the virtual machine rather than virtualising, which provides less performance. QEMU however can be configured to use virtualisation if the grid resource has supporting virtualisation drivers. The performance issues are discussed in Section Chapter 6.

Assuming that the Virtual Machine Work Unit launcher and virtual machine monitor available, the launcher can be executed by simply calling the launcher script. The command-line arguments are parsed by the launcher which checks for any arguments meant for the launcher. Any remaining arguments are assumed to be forwarded to the grid application within the Virtual Machine Work Unit. Full details on the configuration settings of the launcher utility can be found in Appendix D.

The launcher utility allows remote job submissions systems to easily use the Virtual Machine Work Units for grid applications by simply executing the launching utility across the grid resources. Systems like Nimrod, that provide grid middleware for parametric simulations, can utilise the launcher and the Virtual Machine Work Unit can be treated as the grid experiment application. Some examples of generating and executing Virtual Machine Work Units are presented in Section 6.1 **Error! Reference source not found.** using both the Sun Grid Engine and Nimrod.

The launching utility uses the Virtual Machine Work Unit framework discussed in Section 5.4. The launching utility also has some dependencies for allowing the creation of the ISO images and un-packaging of the Virtual Machine Work Unit. A full listing of the dependencies can be found in Appendix E.

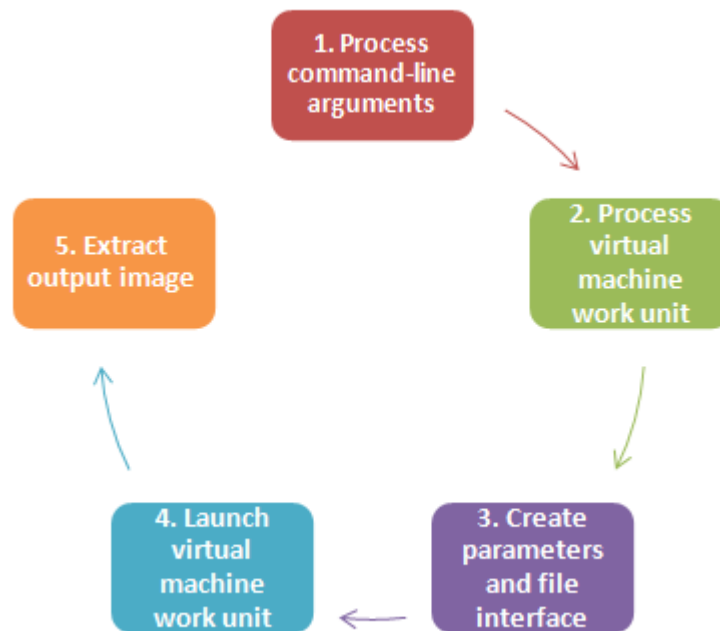


Figure 5.10: Process of the launching utility used to execute Virtual Machine Work Units on a grid resource

The process of launching Virtual Machine Work Units is shown in Figure 5.10. The first step of the launching utility is the processing of the command-line arguments passed to the utility by the remote job execution system. This simply goes through each argument and determines if it belongs to the launching utility or to the grid application. Any arguments for the launcher utility are configuration settings either for the virtual machine monitor, or the generation of the ISO image used to pass the parameters and supporting files to the grid application. Like the packaging utility, the launching utility can also utilise a configuration file.

Once the parameters have been parsed, the Virtual Machine Work Unit specified by one of the parameters is opened. The launcher un-packages the Virtual Machine Work Unit and decompresses it if necessary into a temporary directory created during the initialisation of the launcher. The launcher utility ensures that the base image, application image, and output image exist. Extracting the components to a temporary directory prevents any modifications to the original Virtual Machine Work Unit package promoting reuse; however the e-scientist can specify if the Virtual Machine Work Unit package should be overwritten with the changes that occur during execution. If required the e-scientist can specify external images to be used for each component. This allows easy substitution of Virtual Machine Work Unit components without repackaging all the components again. One such example includes using base images that are stored within the grid infrastructure, and can allow the

removal of the base image from the original Virtual Machine Work Unit reducing any cost of transferring large files.

Once the Virtual Machine Work Unit components have been extracted, the launching utility takes the parameters destined for the grid application and places them into a text file separated by spaces. An ISO image is then created and the parameters file and any other files specified by the launcher arguments are copied on to the ISO image. This ISO image appears to the virtual machine as a CD-ROM and provides the interface for the grid application to receive parameters and supporting files for the execution on the grid resource. If the Virtual Machine Work Unit already contains an ISO image and/or the e-scientist specifies an ISO image, the launching utility attempts to append the files to the existing image.

The virtual machine monitor class in the framework is used to provide the launching mechanisms required for launching the virtual machine on the grid resource. All the virtual machine disks and the ISO image are passed to the virtual machine monitor. The specifications of the virtual machine are provided by the e-scientist and include the virtual machine monitor to be used. The virtual machine monitor class is then used to launch the virtual machine.

At the completion of the grid application within the virtual machine and the shutdown of the virtual machine, the launching utility copies the output image that was modified during execution to the working directory. This can then be copied back to the root node of the grid by grid file transfer mechanisms.

The design of the launcher allows it to be easily extended to support more virtual machine monitors, and provide as much flexibility as possible for the e-scientist. The Virtual Machine Work Unit framework utilised with both the packaging and launching utility will now be discussed.

5.4 Virtual Machine Work Unit Framework

To support the above utilities a basic framework was developed to provide the necessary building blocks needed for dynamically creating and launching Virtual Machine Work Units on the grid.

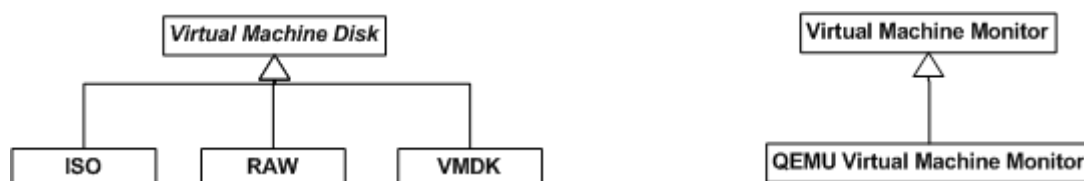


Figure 5.11: Overview of the framework used by the packager and launcher utilities

The framework was developed in Perl and provides a number of classes for virtual machine disks and virtual machine monitors as shown in Figure 5.11. Object oriented design of the framework allowed the utilisation of a defined interface for interacting with virtual machine disks and virtual machine monitors. The framework was designed so that it could easily be extended to support more specific implementations of virtual machine disks and virtual machine monitors.

5.4.1 Virtual Machine Disks

The structure of a Virtual Machine Work Unit is made up of different components with each component excluding the configuration file being virtual machine disks. The ability to provide an abstracted interface for creating and modifying virtual machine images was crucial to allowing the dynamic creation of Virtual Machine Work Units.

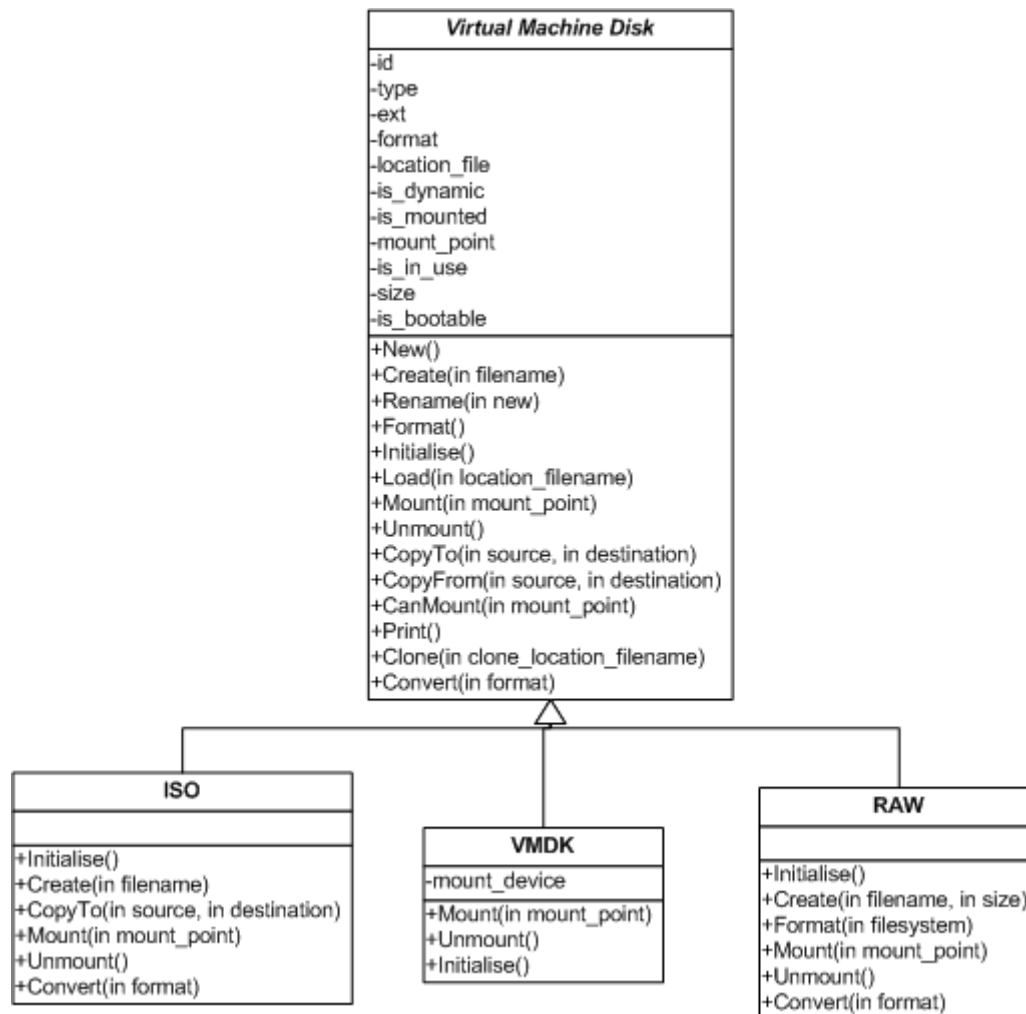


Figure 5.12: Class diagram of the virtual machine disk classes

Thus the development of supporting classes for virtual machine disks was created to allow the packager and launcher to create specific instances of virtual machine disks of varying formats. The development of an abstract virtual machine disk class was implemented with inheriting classes that represent specific virtual machine disk formats as show in Figure 5.12.

Information about each virtual machine is available to the utilities through the framework. Information on the virtual machine disk includes:

- ID – Each virtual machine disk is allocated a unique ID during run-time of the packager and/or launcher.
- Type – The format of the virtual machine disk, in my current implementation this can either be RAW, ISO, and VMDK to a limited extent.
- Extension – The extension of the virtual machine disk. This is used to identify the type of the virtual machine disk when loading from the file system.

- Format – Indicates the format of the file system allocated within the virtual machine disk image.
- Location Filename – This is the location of the virtual machine disk image on the host's file system.
- Dynamic – Indicates if the virtual machine disk is dynamic (grows when space allocated within Guest OS) or if it is static (space allocated at creation time).
- Mounted – Indicates if the current virtual machine disk has been mounted on to the host's file system. Prevents any damage caused by simultaneous access to the virtual machine disk image.
- Mount Point – Indicates the location on the host's file system where the virtual machine disk image has been mounted.
- In Use – Indicates if the virtual machine disk is in use. Prevents any damage caused by simultaneous access to the virtual machine disk image.
- Size – Indicates the file size of the virtual machine disk image on the host's file system.
- Bootable – Indicates if the virtual machine disk contains a bootable operating system or if it simply contains data.

Operations on virtual machine disks is well-defined across most of the sub-types of virtual machine disks, however each of these operations can differ depending on the type. Standard operations on virtual machine disks include the creation, loading, renaming, cloning, converting, formatting, mounting, unmounting, and copying to and from. These operations are provided by the abstracted interface of the virtual machine disk class within the framework.

The creation of virtual machine disks allows the user to create a new virtual machine disk image on the host's file system. The user specifies the required size of the image and the location to create this image. The virtual machine disk image is created on the host's file system, and then the image is formatted to contain a partition table within the virtual machine disk. At the current stage, a virtual machine disk is limited to a single partition which is allocated the maximum space available on the virtual machine disk.

The loading of virtual machine disks allows the user to load previously created virtual machine disks. This operation points the virtual machine disk object to the location of the image on the host's file system.

Support for renaming and cloning existing virtual machine disk images was provided. Due to the nature of virtual machine disk images, virtual machine disk images are files, this was simply file system renaming and copying the images respectively.

Converting virtual machine disks to other formats was designed into the framework and would allow for the conversion of virtual machine disks when dealing with formats not supported by virtual machine disk standards being pushed by major virtualisation vendors.

Formatting virtual machine disks is required to allow the packager and launcher to copy files before the virtual machine is executed. The user can specify the format a given partition on

the virtual machine disk. The function then accesses the virtual machine disk and formats the chosen partition in the given format selected by the user.

The mounting and unmounting of virtual machine disk images allows the user to access the files and folders that are available to the guest operating system and grid application during execution. The user can specify where they want the virtual machine disk image to be mounted.

Finally the ability to copy data to and from the virtual machine disk is supported. This allows the user to simply pass the function the source of any files to be copied across and there relative path on the virtual machine disk.

Using this interface was sufficient for the mechanisms required by both the packaging and launching utilities. However at this stage only limited support for a range of virtual machine disks was provided. The supported virtual machine disk image formats were RAW, VMDK, and ISO. Only limited support for VMDK format was provided at this stage, however the framework could easily be extended to provide complete support for VMDK and other formats.

Each sub-type of the virtual machine disk had some similar operations that were not dependent on the virtual machine disk type, however operations such as creating, mounting, formatting, and copying to and from differed with the implementations of sub-types for the specific virtual machine disk types.

5.4.1.1 RAW Virtual Machine Disk

The RAW virtual machine disk type is simply the raw output of the virtual machine disk without any compression and support for advanced features of virtual machine disks such as dynamic growth. The size of RAW virtual machine disk image corresponds directly to the size of the virtual machine disk, even if space within the virtual machine disk is unused.

Many command tools, most standard Linux distribution tools, were used to support the required operations for the RAW virtual machine disk. These include:

- `qemu-img`[37] – Provided with QEMU virtual machine monitor, `qemu-img` is a utility that allows the creation of virtual machine disk images. It supports various virtual machine disk types such as QCOW, QCOW2, VMDK, and RAW.
- `losetup`[71] – Provided with standard GNU tools, `losetup` is a utility for setting up loopback devices on the host system. It takes an image and allows it to be accessed through a device rather than through the file system.
- `fdisk`[72] – Fixed disk is provided with standard GNU disk tools and allows the creation and modification of partition tables of disk devices.
- `mkfs`[73] – Make file system is provided with standard GNU disk tools, however it is usually a wrapper utility that takes a file system format and launches the appropriate formatting tool.
- `mount`[74] – Standard Linux command for mounting a device and/or file on to the file system.
- `umount`[75] – Standard Linux command for unmounting devices and/or files mounted to the file system.

The usage of these tools will be detailed in virtual machine disk type specific operations.

The creation of RAW virtual machine disks utilised the tool developed alongside the QEMU virtual machine emulator. The image creation utility known as “qemu-img” allowed the creation of virtual machine disks of various types. The operations by this class wrap around “qemu-img” and allow the user to specify the required size of the virtual machine disk image and the location of the virtual machine disk image. “qemu-img” is launched by the function through the shell and the virtual machine disk image is created on the host file system in the location defined by the user.

However to provide the necessary support for utilising this virtual machine disk image, the virtual machine disk required to be partitioned to ensure compatibility for supporting data copied to this disk and when the virtual machine disk is accessed directly within the virtual machine. “fdisk” was required to initialise the virtual machine disk and to create the necessary partition table information within the virtual machine disk. The virtual machine image also needed to be setup as a loopback device using “losetup” to allow “fdisk” to access and create the partition table. The creation operation feeds “fdisk” a script to automatically create a single partition on the virtual machine disk. The dimensions of the virtual machine disk such as the number of cylinders, number of heads, number of sectors, and sector size are automatically calculated by “fdisk”.

The formatting of partitions within virtual machine disks is also provided by the framework, however with the current implementation leads to some issues with the portability of virtual machines depending on the host operating system. Using “mkfs”, the virtual machine disk can be formatted into various file systems supported by the host operating system. The implementation was developed within a Linux distribution and such had access to formats such as ext2, ext3, ntfs, vfat, etc. Most operating systems support one of these file system formats. Extending this support is critical in improving the Virtual Machine Work Unit framework. When formatting the disk, like in creation of the partition tables the virtual machine disk image needs to be setup as a loopback device pointing directly to the partition. However because we are only formatting a partition not the entire virtual machine disk we need to calculate the offset of the partition. This is calculated by retrieving the dimensions of the virtual machine disk provided by “fdisk”. This information is then used to calculate the actual offset of the partition and used when formatting the partition.

Mounting RAW virtual machine disks again follows the steps required to setup the virtual machine disk image as a loopback device and calculate the partition offset. The format of the partition file system is also required for mounting the virtual machine disk on to the file system.

Copying to and from the RAW virtual machine disk was supported using the file system copying mechanisms. This involved mounting the virtual machine disk image to a temporary location and copying data to and from the virtual machine disk. Once completed the virtual machine disk image was unmounted.

5.4.1.2 ISO Virtual Machine Disk

The launching utility uses CD-ROM image generation to allow the passing of parameters and files at the instantiation of Virtual Machine Work Units on grid resources. The creation and copying of data to and from these generated CD-ROM images without root access was critical, as the launching utility was designed to run on existing grid infrastructure and without the need for installing root privilege level components by grid administrators.

CD-ROM images are represented by ISO images and follow the ISO standards for ISO9660. This is to ensure the compatibility of such images across a wide range of platforms and operating system architectures.

Standard Linux utilities were used to facilitate the creation of ISO virtual machine disks and these include:

- `mkisofs`[76] – Part of the `cdrtools` package of GNU, `mkisofs` is used to create ISO file systems and images.
- `growisofs`[77] – Part of the `cdrtools` package of GNU, `growisofs` wraps around `mkisofs` and allows the adding and removal of files from an existing multi-session ISO image.
- `mount`[74] – Standard Linux command for mounting a device and/or file on to the file system.
- `umount`[75] – Standard Linux command for unmounting devices and/or files mounted to the file system.

The usage of these tools will be detailed in virtual machine disk type specific operations.

The method for creating ISO virtual machine disks made use of “`mkisofs`” which would create an ISO image containing the files and folders initially passed to it. However, to follow the interface rules as per other virtual machine disks, the creation operation would create an empty virtual machine disk. “`mkisofs`” does not directly provide a method for creating empty ISO images however running this command within an empty directory and passing the current directory (.) to “`mkisofs`” allows the creation of empty ISO images. No partitioning or formatting is required as an ISO image adheres to the ISO9660 format and the image is directly interpreted by the virtual machine monitor to appear as a CDROM device to the virtual machine and guest operating system.

Copying data to the ISO image avoids requiring root access by using the utility “`growisofs`”, which allows files to be copied to the ISO image. Using graph point functionality provided by “`growisofs`”, the framework wraps around this utility and allows the user to copy files to a specific location within the ISO image. Unlike previous copying techniques, the ISO virtual machine disk does not need to be mounted to copy files to; however retrieving files from the ISO virtual machine disk does require mounting, though at this stage the functionality for retrieving files from the ISO image is not critical.

Mounting the ISO virtual machine disk image can be achieved by using the `mount` command, however no loop device needs to be setup as this functionality can be provided directly by `mount`.

5.4.2 Virtual Machine Monitors

Providing support to launch virtual machine monitors was required for intercepting and allowing the use of ISO images to pass parameters and files to the grid application when the Virtual Machine Work Unit is instantiated. It was also needed to provide temporary interface to the virtual machine monitors installed on the grid resource and or in the case of no existing virtual machine monitor allow the usage of a portable virtual machine monitor such as QEMU.

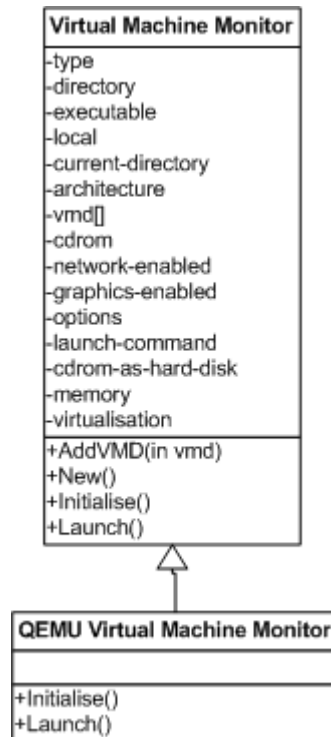


Figure 5.13: Support for virtual machine monitors

Thus the development of supporting class for virtual machine monitors was created to allow the launcher to utilise and support many virtual machine monitor implementations. The development of an abstract virtual machine monitor class was implemented with inheriting classes that represent specific virtual machine monitor implementations as show in Figure 5.13.

Settings and information on the virtual machine monitor are used to provide flexibility for launching a range of virtual machine monitors. This information includes:

- Type – The virtual machine monitor implementation.
- Directory – The location of the directory that contains the virtual machine monitor.
- Executable – The filename of the virtual machine monitor executable responsible for launching virtual machines.
- Local – Flags whether the virtual machine is installed locally on the grid resource. Indicates if the directory information should be used or if executable is in path environment variable.
- Current Directory – The current directory of the environment.
- Architecture – The architecture of the virtual machine to be emulated or virtualised.

- Virtual Machine Disks – An array of virtual machine disks to be used by the virtual machine.
- CD-ROM – The virtual machine disk to be used as a CD-ROM by the virtual machine.
- Network – Indicates if the network interface is to be enabled within the virtual machine. At this stage the Virtual Machine Work Units do not support network connectivity for parallel applications.
- Graphics – Indicates if the graphics output from the virtual machine monitor is enabled, if not the virtual machine will run in the background. Used for testing Virtual Machine Work Unit environments before launching onto the grid.
- Options – And additional command-line arguments to be passed to the virtual machine monitor.
- Launch Command – The command used to launch the virtual machine on the virtual machine monitor. This is usually generated when using the launching operation however users can customise.
- CD-ROM as Hard Disk – Indicates if the CD-ROM image should be passed to the virtual machine as a hard disk. Used to support guest operating systems that have trouble reading generated ISO images on emulated CD-ROM devices.
- Memory – Indicates the memory size of the virtual machine. By default this is 512MB, however the user can allocate the required memory size. This will be redundant once OVF configuration file is completely implemented for Virtual Machine Work Units.
- Virtualisation – Indicates if the virtual machine monitor is to emulate the virtual machine or virtualise the virtual machine. This is discussed in-depth in Section 3.2.

To provide maximum flexibility, the base virtual machine monitor class was designed such it could be utilised by providing access to the launching command variable. This is used when launching the virtual machine on the virtual machine monitor and such the user can specify exactly how the Virtual Machine Work Unit is launched on the grid.

The main operations provided by the virtual machine monitor class are fairly minimal but enough to provide flexible support needed for the wide-range of virtual machine monitors. Virtual machines that will be launched by the virtual machine monitor can be added by referencing virtual machine disk objects from within the framework. Launching the virtual machine is as simple as calling the launch method.

At this stage support for the virtual machine emulator QEMU is only implemented, however the scaffolding is there to support other virtual machine monitors.

5.4.2.1 QEMU Virtual Machine Monitor

The QEMU implementation of the virtual machine monitor class was provided as the basic support for launching Virtual Machine Work Units. The construction of the launching command will be discussed to show the different ways QEMU is used.

QEMU provides support for up to four virtual machine disks using `-hda` through to `-hdd`; however when using the `-cdrom` option the usage of `-hdc` is invalid. Due to the nature of Virtual Machine Work Units, the third argument `-hdc` is not utilised and reserved always for CD-ROM images. The flag `cdrom-as-hard-disk` within the virtual machine monitor class is

responsible for determining if the CD-ROM image is passed either through the `-hdc` option or the `-cdrom` option.

Networking is enabled by default by QEMU however in the case when the `network-enabled` is false the option `-no-network` is passed to QEMU to disable networking emulation. QEMU normally provides network access through NAT, and also provides other means of networking. This is not utilised by Virtual Machine Work Units at this stage.

When QEMU is not installed locally within the grid resource it requires the location of the BIOS which are represented by files that come with QEMU distribution. This is passed to the QEMU executable in this case.

Graphic output is disabled when required in QEMU by providing the `-no-graphics`. QEMU also provides other mechanisms with interfacing with the Virtual Machine Work Unit such as providing a VNC interface, however support for this is not utilised at this stage by the Virtual Machine Work Units though in the future this could provide an interface to viewing Virtual Machine Work Units during execution on grid resources.

Memory size of the virtual machine is predefined to be 128MB by QEMU, however most grid applications require more memory than this. By default the framework sets this to 512MB however this can be modified by the e-scientist if required.

QEMU provides support for virtualisation using the `kqemu` accelerator driver if installed within the grid resource. By default, if the driver exists and user access rights are granted to this device, QEMU makes use of virtualisation for non-kernel related instructions. QEMU also provides virtualisation for kernel related instructions if the `-kernel-kqemu` is passed to QEMU. Virtualisation can only be utilised when the virtual machine is the same architecture as the grid resource, otherwise emulation is enabled.

Support for architectures that differ from the host are provided using the `qemu-[architecture]` executables provided with QEMU. The architecture supported by the virtual machine emulator can be selected by changing the architecture attribute. At this stage there is no filtering to ensure the architecture specified is supported by that particular virtual machine monitor.

Another issue with QEMU is that it is designed and executes like a service rather than a command and stays active even after the halting of the virtual machine. This prevents the virtual machine launcher from recognising when the grid application within it has completed. As such the option `-no-reboot` is passed to QEMU that implements the termination of the QEMU process if the halting or reboot signal is sent from the virtual machine.

Chapter 6 Demonstration and Performance Considerations

Virtual Machine Work Units remove the constraints in developing and deploying grid applications, and provide the ability to execute grid applications within their own architectural and operating system requirements. However, utilising virtual machines comes with disadvantages including the complexity of creating execution environments, the overheads of emulating and virtualising a complete computing platform, and the large file sizes that virtual machine disks encumber. As such it is important to test the feasibility of Virtual Machine Work Units.

To test the feasibility of Virtual Machine Work Units we looked at two important aspects:

- The **useability** of using Virtual Machine Work Unit as a method of encapsulating grid applications. In Section 6.1 we demonstrate two different grid applications and how they can be encapsulated within a Virtual Machine Work Unit. This includes looking at how they can be launched on existing grid infrastructure using grid middleware such as the Sun Grid Engine and Nimrod.
- The **performance** metrics of Virtual Machine Work Units on existing grid infrastructure. In Section 6.2 we look at the metrics involved with creating and executing Virtual Machine Work Units. It should be noted that research on the performance of virtual machines is outside the scope of this thesis. More information on the performance of virtualisation in high-performance computing is discussed in Section 3.4.1; this also provides references to further information.

The feasibility of Virtual Machine Work Units was tested using the following resources:

- *Single workstation (Development environment)*

The single workstation was available for the generation and execution of Virtual Machine Work Units. This workstation was also used to test the Virtual Machine Work Units by enabling graphics when in execution to observe interactions and behaviours of the grid application within the virtual machine.

The single workstation contains an Intel Dual Core T2060 (1.66GHZ) with 1.5GB of memory. The host operating system is Ubuntu 8.04 (Linux Kernel 2.6.X) and QEMU (0.9.1) was used in launching the Virtual Machine Work Units with support for virtualisation using the KQEMU (1.4.0pre11) drivers.

- *Grid (Access to single computing cluster)*

The Grid was provided as part of the Message Lab infrastructure. The grid consists of four computing clusters and is known as the Enterprise Grid and is provided by Monash University (Melbourne, Australia), Royal Melbourne Institute of Technology (RMIT) (Melbourne, Australia), Deakin University (Melbourne, Australia), and Queensland University of Technology (QUT) (Brisbane, Australia). Access was provided to a single cluster within this computing grid and was provided by Monash University. The computing cluster is referred to as EAST.

The EAST computing cluster consists of twenty dual CPU quad core (160 cores) Intel Xeon E5310 (1.66GHz) computers with 8Gb RAM interconnected by InfiniBand 4x SDR (~10Gb/s) and GigE, and sixteen dual CPU dual core (64 cores) Intel Xeon 5160 (3GHz) computers with 8Gb RAM and GigE interconnect. More information on the EAST computing cluster can be found here[78]. Each grid resource runs CentOS Linux (Kernel Version 2.4.X).

The EAST computing cluster resource can be utilised by the Sun Grid Engine that is installed across the computing cluster, or using Nimrod/G which uses Globus Toolkit across the computing cluster. The remote job submission systems of this grid middleware were used for executing Virtual Machine Work Units across grid resources.

QEMU was installed on the grid resources and virtualisation through KQEMU was also installed on all grid resources. However, initially QEMU was copied in separately and used to launch Virtual Machine Work Units. The QEMU version installed across the grid resources was version 0.9.0 and the KQEMU driver was version 1.3.0pre11. Both QEMU and KQEMU were compiled for the Linux 64-bit operating system.

The useability of Virtual Machine Work Units is presented in the following section as a demonstration of two different grid applications that are encapsulated into Virtual Machine Work Units and launched on to existing grid infrastructure.

6.1 Demonstration of Virtual Machine Work Units

The first demonstration presents a grid application used to calculate the Mandelbrot Set. The process of calculating each point with the set is computationally expensive; however each point can be computed independently and is well suited for parallel computing.

The second demonstration presents a grid application used for simulating the phasing of macromolecular crystal structures. The simulation is computationally expensive and each simulation can take hours to days to run depending on the input.

6.1.1 Grid Application 1- Mandelbrot Set

The Mandelbrot set is the set of points within the complex plane that edges form a fractal image[79]. This set is obtained by applying a complex quadratic polynomial function repeatedly to the two dimensional set comprising of complex numbers[79]. The visual representation of the Mandelbrot set can be seen in Figure 6.1.

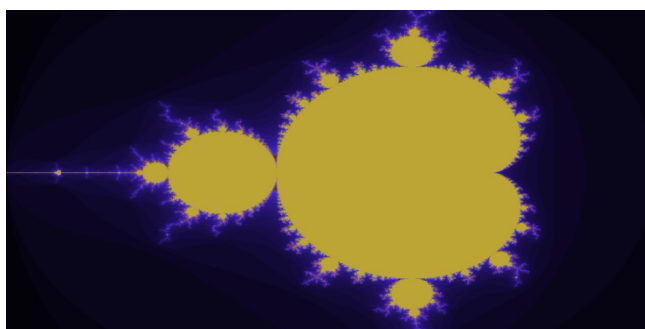


Figure 6.1: Visualisation of the Mandelbrot set

The Mandelbrot set is computationally intensive for large sets of complex numbers and the number of iterations that are used to calculate each point. Each point is calculated independently and does not rely on information from other points. This provides a suitable mechanism to apply parallel computing by calculating points in parallel reducing the time required to calculate the entire Mandelbrot set.

In this example, the grid application is used to calculate the values for a given set of points. The grid application outputs the calculated values which are later on combined with other values by another Mandelbrot utility to create the visualisation as seen above.

The application is implemented in C++ and does not rely on any software dependencies. This example is provided to show the simplicity of using the packaging and launching mechanisms developed. The application is launched using the Sun Grid Engine as discussed in Section 2.2.1.

The experiment example is the calculation of the Mandelbrot set for the domain -2.0 to 1.0 in the real component of the complex plane, and the range -1.0 to 1.0 for the imaginary component of the complex plane. The resulting image is specified to be 1600 x 800 pixels and each pixel is to be calculated over 100 iterations.

The work will be broken down into separate jobs where each job is responsible for calculating a subset of the Mandelbrot set (A portion of the image). Each job represents 100 rows of pixels within the image. A job corresponds to an execution of the Mandelbrot Virtual Machine Work Unit.

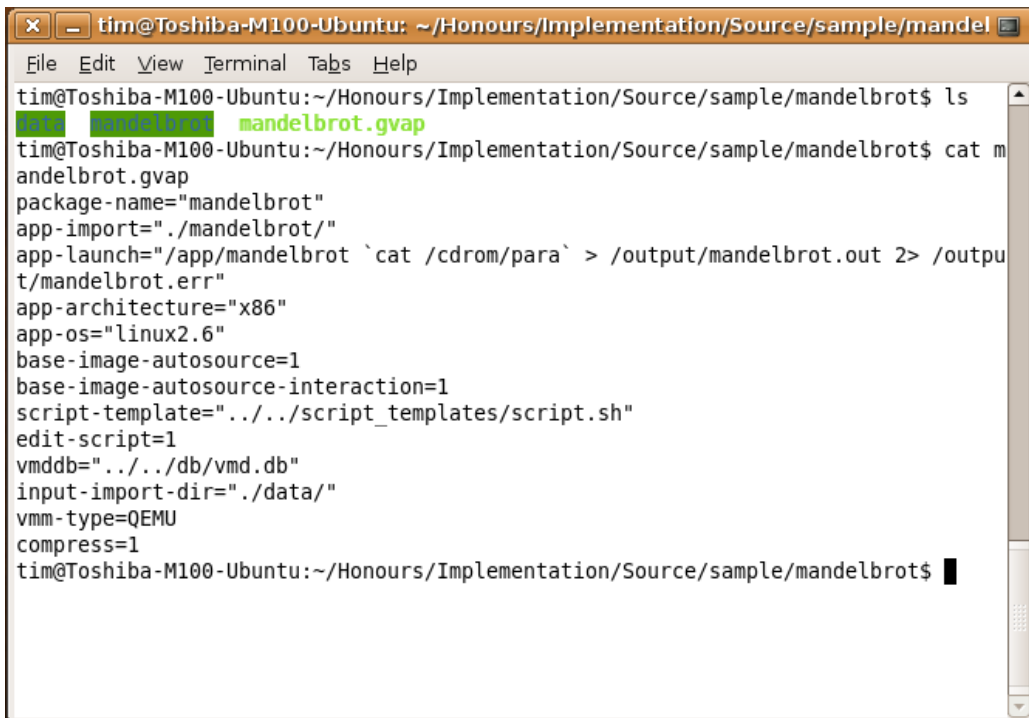
The jobs will be launched using a script that uses the Sun Grid Engine job submission utilities. This script will create the parameters for each job, and then submit it to the grid job queue. The resulting images will then be returned to a single directory where each output name is uniquely named for each job. These will then be combined into a single TAR file and returned to the workstation where the contents of each output image will be outputted to a single directory.

This example will be broken down into two sections; the packaging of the Mandelbrot grid application into the Virtual Machine Work Unit, and the execution of the Mandelbrot Virtual Machine Work Unit across the grid.

6.1.1.1 Generation of the Mandelbrot Virtual Machine Work Unit

As previously mentioned, the Mandelbrot grid application is a simple executable that needs to be executed with parameters passed in at run-time. The execution environment required is very basic and is designed to run within an "x86" architecture virtual machine, and on a "linux2.6" based operating system.

A configuration file was created to represent these requirements needed by the Mandelbrot grid application, as well as information on the location of the application executable and launching command.

A terminal window titled "tim@Toshiba-M100-Ubuntu: ~/Honours/Implementation/Source/sample/mandel" displays the contents of a configuration script. The user has run 'ls' and 'cat mandelbrot.gvap'. The script content is as follows:

```
tim@Toshiba-M100-Ubuntu:~/Honours/Implementation/Source/sample/mandelbrot$ ls
mandelbrot  mandelbrot.gvap
tim@Toshiba-M100-Ubuntu:~/Honours/Implementation/Source/sample/mandelbrot$ cat mandelbrot.gvap
package-name="mandelbrot"
app-import="./mandelbrot/"
app-launch="/app/mandelbrot `cat /cdrom/para` > /output/mandelbrot.out 2> /output/mandelbrot.err"
app-architecture="x86"
app-os="linux2.6"
base-image-autosource=1
base-image-autosource-interaction=1
script-template="../../script_templates/script.sh"
edit-script=1
vmddb="../../db/vmd.db"
input-import-dir="/data/"
vmm-type=QEMU
compress=1
tim@Toshiba-M100-Ubuntu:~/Honours/Implementation/Source/sample/mandelbrot$
```

Figure 6.2: Configuration script for the Mandelbrot grid application

Figure 6.2 shows the configuration script used to represent the Mandelbrot grid application. We assume that the location of the virtual machine disk database is included with the packaging utility. The launching command allows the parameter file passed in through the CD-ROM at run-time to be passed to the Mandelbrot grid application by using ``cat /cdrom/param``. This concatenates the contents of the parameter file to the command-line arguments of the Mandelbrot grid application. Other notable settings are that the base image will be auto-sourced from the virtual machine disk database, the script template is provided, and the script will be edited during the packaging process.

```

tim@Toshiba-M100-Ubuntu: ~/Honours/Implementation/Source/sample/mandel
File Edit View Terminal Tabs Help
alAppliancePackager.pl line 278.
virtual-machine-disk-database =
vm-interaction = 0
vmm-architecture =
vmm-directory =
vmm-executable =
vmm-local = 0
vmm-type = QEMU
=== End Configuration ===
Configuring application... Complete!
Loading Virtual Machine Disk Database... Complete!
vmd_id|vmd_name|vmd_description|vmd_location|vmd_architecture|vmd_guest_os|vmd_f
ilesystem|vmd_input_mount_device|vmd_output_mount_device|vmd_cdrom_mount_device|
vmd_script_template|vmd_date_added
1|Ubuntu 7.04|U:auto P:auto|/home/tim/Honours/Implementation/Source/images/x86_u
buntu_7_04.img|x86|linux2.6|ext2|/dev/sdb1|/dev/sdd1|/dev/cdrom|linux_script.sh|
2008-10-15 11:06:08
2|Gentoo 2006.1|U:root P:gentoo|/home/tim/Honours/Implementation/Source/images/x
86_gentoo_2006_1.img|x86|linux2.6|ext2|/dev/hdb1|/dev/hdd1|/dev/cdrom|linux_scri
pt.sh|2008-10-15 11:06:45
3|Damn Small Linux 4.4.4|U:root P:|/home/tim/Honours/Implementation/Source/image
s/x86_dsl_4_4_4.img|x86|linux2.6|ext2|/dev/hdb1|/dev/hdd1|/dev/hdc|linux_script.
sh|2008-10-15 11:08:51
Enter ID of selected VMD? █

```

Figure 6.3: Base images that can be used to encapsulate the Mandelbrot grid application

The packager utility is launched with the following command: “launcher.pl –config mandelbrot.cfg”. Once it the launcher begins to execute it presents a list of base images that can be used by the Mandelbrot grid application and is shown in Figure 6.3. In this example we select the “Damn Small Linux” base image due to its relatively small size.

```

tim@Toshiba-M100-Ubuntu: ~/Honours/Implementation/Source/sample/mandel
File Edit View Terminal Tabs Help
#!/bin/sh
# Mount output image
mkdir /output
mount -t ext2 /dev/hdd1 /output
# Mount ISO and copy contents to input directory if present
mkdir /cdrom
mount /dev/hdc /cdrom
# Go to application working directory
cd /output
# Instance specific commands
gcc -o /app/mandelbrot /app/mandelbrot.c
# Launch application
/app/mandelbrot `cat /cdrom/para` > /output/mandelbrot.out 2> /output/mandelbrot
.err
# Instance Post specific commands

# Shut down VM
reboot
~
~
~
~

```

Figure 6.4: Generated virtual machine auto-run script for the Mandelbrot Virtual Machine Work Unit

The packaging utility continues to execute until it reaches the completion of the virtual machine auto-run script generation. VIM text editor is loaded and the auto-run script used within the Virtual Machine Work Unit can be edited as shown in Figure 6.4. To ensure that the application is fully compliant with the execution environment, we add the extra line ahead of the launching command to compile the C++ source code of the Mandelbrot grid application. This is not always necessary as we could execute and modify the master Virtual Machine Work Unit and compile the application then, however due to its relative small size, the overhead is minimal for compiling each time it is launched.

The changes to the script are saved and the packaging utility continues executing until the complete Virtual Machine Work Unit is packaged. The Virtual Machine Work Unit is now ready to be transferred and executed on the grid.

6.1.1.2 Launching of the Mandelbrot Virtual Machine Work Unit

Each Virtual Machine Work Unit execution represents a job to calculate rows of pixels for the Mandelbrot set. All the output images are then processed to retrieve the results which are then combined to form the Mandelbrot set visualisation.

The launcher and QEMU are located in a global home directory accessible by each grid resource. The Virtual Machine Work Unit is also located in this directory.

A script was written to automatically calculate the job parameters and launch the Virtual Machine Work Units through the Sun Grid Engine job submission utility. This was executed from the root node of the grid.

Once all jobs are completed, the uniquely named output images are transferred back to the workstation. Because each output file within the virtual machine disk is uniquely named, each virtual machine disk can be outputted to a single directory. Another utility implemented was used that takes a list of virtual machine disks, and extracts their contents to a specified directory.

The combining Mandelbrot executable takes a list of files, and then combines these files to create the Mandelbrot set visualisation. This is simply the list of files that were extracted from all the virtual machine disk images.

The resulting image is achieved and we have successfully executed an experiment using Virtual Machine Work Units. The next demonstration provides a similar example however the grid application doesn't utilise command-line arguments and uses supporting experiment files instead as its input.

6.1.2 Grid Application 2- Phaser

Phaser is a grid application developed by McCoy et al [80] and is a C++ implementation of a software application that simulates the phasing of macromolecular crystal structures. It supports a wide range of platforms.

Simulations can range from minutes to days depending on the input provided to Phaser. Unlike the Mandelbrot set grid application, phaser takes input through data files, and a scripting file that is directed to its standard input. The output of the phaser is then stored in another text file, and its standard out provides information on the simulation. All this output needs to be collected for each simulation.

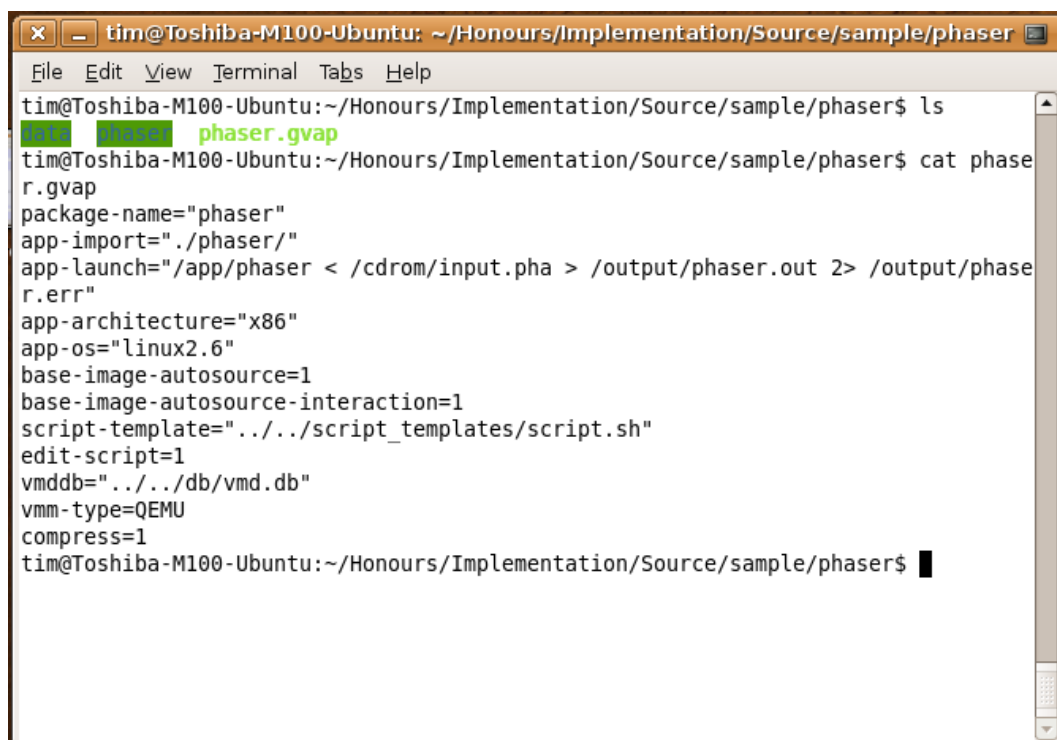
The simulations differ depending on these files passed, however it is the responsibility of the e-scientist to compile these files before using the simulation across the grid.

In this example we use Nimrod, a tool for parametric simulations across grids. This example will be broken down into two sections; the packaging of the Phaser grid application into the Virtual Machine Work Unit, and the execution of the Phaser Virtual Machine Work Unit across the grid. The aim of this example is to demonstrate the usage of grid applications that utilise files as input rather than parameters.

6.1.2.1 Generation of the Phaser Virtual Machine Work Unit

As previously mentioned, the Phaser grid application is an executable that needs to be executed with supporting files and input directed through standard input stream passed in at run-time. The execution environment required is very basic and is designed to run within an "x86" architecture virtual machine, and on a "linux2.6" based operating system.

A configuration file was created to represent these requirements needed by the Phaser grid application, as well as information on the location of the application executable and launching command.



```
tim@Toshiba-M100-Ubuntu: ~/Honours/Implementation/Source/sample/phaser
File Edit View Terminal Tabs Help
tim@Toshiba-M100-Ubuntu:~/Honours/Implementation/Source/sample/phaser$ ls
phaser phaser.gvap
tim@Toshiba-M100-Ubuntu:~/Honours/Implementation/Source/sample/phaser$ cat phaser.gvap
package-name="phaser"
app-import="./phaser/"
app-launch="/app/phaser < /cdrom/input.pha > /output/phaser.out 2> /output/phaser.err"
app-architecture="x86"
app-os="linux2.6"
base-image-autosource=1
base-image-autosource-interaction=1
script-template="../../script_templates/script.sh"
edit-script=1
vmddb="../../db/vmd.db"
vmm-type=QEMU
compress=1
tim@Toshiba-M100-Ubuntu:~/Honours/Implementation/Source/sample/phaser$
```

Figure 6.5: Configuration script for the Phaser grid application

Figure 6.5 shows the configuration script used to represent the Phaser grid application. We assume that the location of the virtual machine disk database is included with the packaging utility. The launching command allows the standard input to be passed in through the CD-ROM using a file at run-time to be passed to the Phaser grid application by using input redirection. The standard output and standard error are redirected to the output image. Other notable settings are that the base image will be auto-sourced from the virtual machine disk database, the script template is provided, and the script will be edited during the packaging process.

```

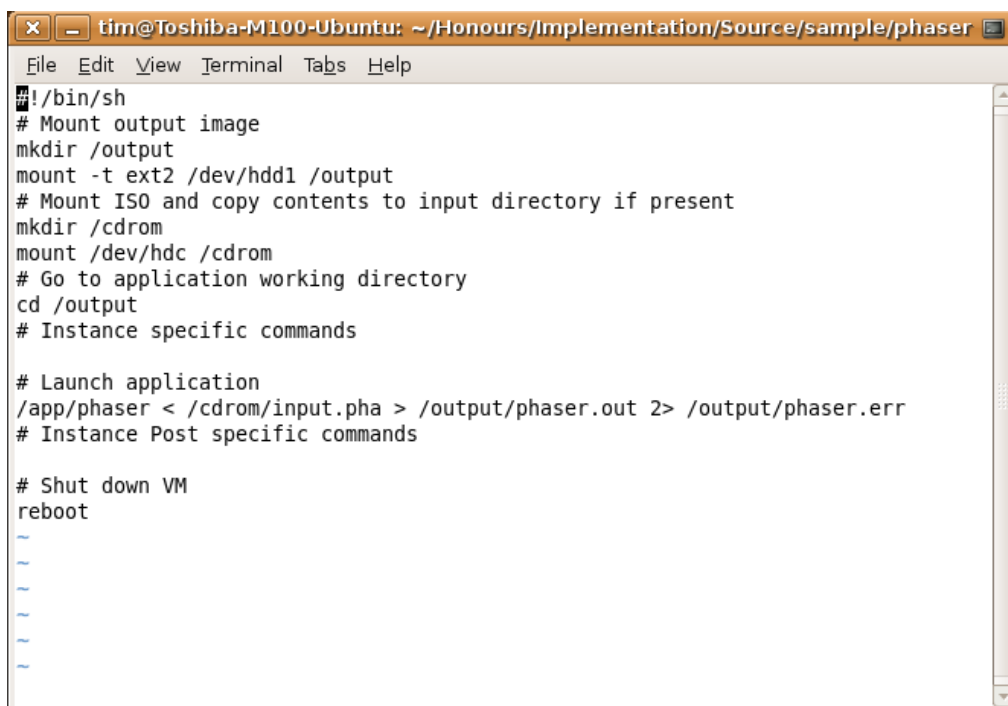
alAppliancePackager.pl line 278.
virtual-machine-disk-database =
vm-interaction = 0
vmm-architecture =
vmm-directory =
vmm-executable =
vmm-local = 0
vmm-type = QEMU
=== End Configuration ===
Configuring application... Complete!
Loading Virtual Machine Disk Database... Complete!
vmd_id|vmd_name|vmd_description|vmd_location|vmd_architecture|vmd_guest_os|vmd_f
ilesystem|vmd_input_mount_device|vmd_output_mount_device|vmd_cdrom_mount_device|
vmd_script_template|vmd_date added
1|Ubuntu 7.04|U:auto P:auto|/home/tim/Honours/Implementation/Source/images/x86_u
buntu_7_04.img|x86|linux2.6|ext2|/dev/sdb1|/dev/sdd1|/dev/cdrom|linux_script.sh|
2008-10-15 11:06:08
2|Gentoo 2006.1|U:root P:gentoo|/home/tim/Honours/Implementation/Source/images/x
86_gentoo_2006_1.img|x86|linux2.6|ext2|/dev/hdb1|/dev/hdd1|/dev/cdrom|linux_scri
pt.sh|2008-10-15 11:06:45
3|Damn Small Linux 4.4.4|U:root P:|/home/tim/Honours/Implementation/Source/image
s/x86_dsl_4_4_4.img|x86|linux2.6|ext2|/dev/hdb1|/dev/hdd1|/dev/hdc|linux_script.
sh|2008-10-15 11:08:51
Enter ID of selected VMD? █

```

Figure 6.6: Base images that can be used to encapsulate the Phaser grid application

The packager utility is launched with the following command: “launcher.pl –config phaser.cfg”. Once it the launcher begins to execute it presents a list of base images that can be used by the Phaser grid application and is shown in Figure 6.6. In this example we select the “Damn Small Linux” base image due to its relatively small size.

The packaging utility continues to execute until it reaches the completion of the virtual machine auto-run script generation. VIM text editor is loaded and the auto-run script used within the Virtual Machine Work Unit can be edited as shown in Figure 6.7.



```
tim@Toshiba-M100-Ubuntu: ~/Honours/Implementation/Source/sample/phaser
File Edit View Terminal Tabs Help
~/bin/sh
# Mount output image
mkdir /output
mount -t ext2 /dev/hdd1 /output
# Mount ISO and copy contents to input directory if present
mkdir /cdrom
mount /dev/hdc /cdrom
# Go to application working directory
cd /output
# Instance specific commands

# Launch application
/app/phaser < /cdrom/input.pha > /output/phaser.out 2> /output/phaser.err
# Instance Post specific commands

# Shut down VM
reboot
~
~
~
~
~
```

Figure 6.7: Generated virtual machine auto-run script for the Phaser Virtual Machine Work Unit

The changes to the script are saved and the packaging utility continues executing until the complete Virtual Machine Work Unit is packaged. The Virtual Machine Work Unit is now ready to be transferred and executed on the grid.

6.1.2.2 Launching of the Phaser Virtual Machine Work Unit

The e-scientist is responsible for preparing the data files required to be used by the Phaser grid application. This is no different if the application was to be launched natively across the grid. Three files are required for each simulation and include:

- File that represents the standard input that will be used by the Phaser executable.
- Two supporting files that are used in the simulation for the phasing of macromolecular structures.

```
1  MODE MR_AUTO
2  HKLin /cdrom/phaser.mtz
3  LABIN F=F_P321_295 SIGF=SIGF_P321_295
4  TITLE PxGrid
5  SGALTERNATIVE TEST P321
6  COMPOSITION PROTEIN MW 50000 NUMBER 2
7  ENSEMBLE pdb PDBFILE /cdrom/phaser.pdb IDENTITY 50
8  SEARCH ENSEMBLE pdb NUMBER 1
9  PACK 10
10 ROOT /cdrom/phaser.sum
11
```

Figure 6.8: Experiment file that is directed into the Phaser grid application within the virtual machine

Phaser input takes information on what supporting files should be used, and the results of the simulation should be outputted to. Figure 6.8 shows the standard input provided to phaser for one simulation. From within the virtual machine, this file is located at

“/cdrom/input.pha”. Within this file the two files used by the simulation are referred to as “/cdrom/phaser.mtz” and “/cdrom/phaser.pdb”. The results of the simulation are outputted to “/cdrom/phaser.sum”. Utilising the naming convention of files, this Virtual Machine Work Unit can be reused for other phaser experiments without repackaging the entire Virtual Machine Work Unit.

```
1
2 parameter gva text default "phaser.ova.gz";
3 parameter index integer range from 1 to 20 step 1;
4
5 task main
6     copy ${gva} node:..
7     copy ${index}.pha node:input.pha
8     copy ${index}.mtz node:phaser.mtz
9     copy ${index}.pdb node:phaser.pdb
10    node:execute ${HOME}/gva/GridVirtualApplianceLauncher.pl -gva
    ${gva} -compressed -vmm-type QEMU -cdrom-as-hard-disk -files
    "input.pha,phaser.mtz,phaser.pdb" > gval.out 2> gval.err
11    copy node:output.img output_${index}.img
12    copy node:gval.out gval_${index}.out
13    copy node:gval.err gval_${index}.err
14 endtask
```

Figure 6.9: Nimrod plan file for the Phaser experiment

Nimrod was used to provide the parameter sweeping needed for launching many parallel phaser simulations. The files for each simulation are conventionally named as “<index>.pha”, “<index>.mtz”, and “<index>.pdb”. Figure 6.9 shows the plan file used to coordinate the experiment with Nimrod. Each job represents a single simulation which is provided with a unique index which is used to distinguish the experiment files used for each simulation. These files are copied to the grid resource and are generically named. The launching utility is used to execute the Virtual Machine Work Unit across the grid using QEMU. The experiment files are passed in using this interface. The resulting output image of each simulation is then copied back to the root node of the grid. These output images can now be processed to check the results of each phasing simulation.

Both demonstrations provide the necessary basis on the effort required by e-scientists to use Virtual Machine Work Units; however it is important to realise the cost of removing traditional grid application constraints. The next section details the implementation metrics of Virtual Machine Work Units and the performance considerations that need to be taken into account on using Virtual Machine Work Units on existing grid infrastructure.

6.2 Virtual Machine Work Unit Metrics

Virtual Machine Work Units utilise virtual machines to provide the e-scientist with ability to define the execution environment of their grid application. The utilisation of virtual machines leads to overheads that can lead to performance costs when executing grid applications. The overheads extend to both the size of deploying grid applications and execution of grid applications.

The following sections analyse and present the performance feasibility of Virtual Machine Work Units. This includes looking at the instantiation performance metrics of Virtual Machine Work Units and then finally the virtualisation performance of using Virtual Machine Work Units.

6.2.1 Virtual Machine Work Unit Instantiation Performance Metrics

Virtual machines are complete computing platforms and such a lot of overhead is used to support the execution environment used by the encapsulated grid application within the Virtual Machine Work Unit. Within the virtual machine, the entire process of booting the system and loading of services within the guest operating system takes place before the grid application is launched. Once completed, the entire process of shutting down takes place including the termination of services and the unmounting of file systems.

As such the time that the Virtual Machine Work Unit takes to execute is equal to:

$$\text{Boot Time} + \text{Application Execution} + \text{Shutdown Time} = \text{Total Time}$$

Proportionately the boot time and shutdown time is not a large component of the total execution time, however decreasing this time should reduce the amount of wasted computation if such a commodity has costs associated with it.

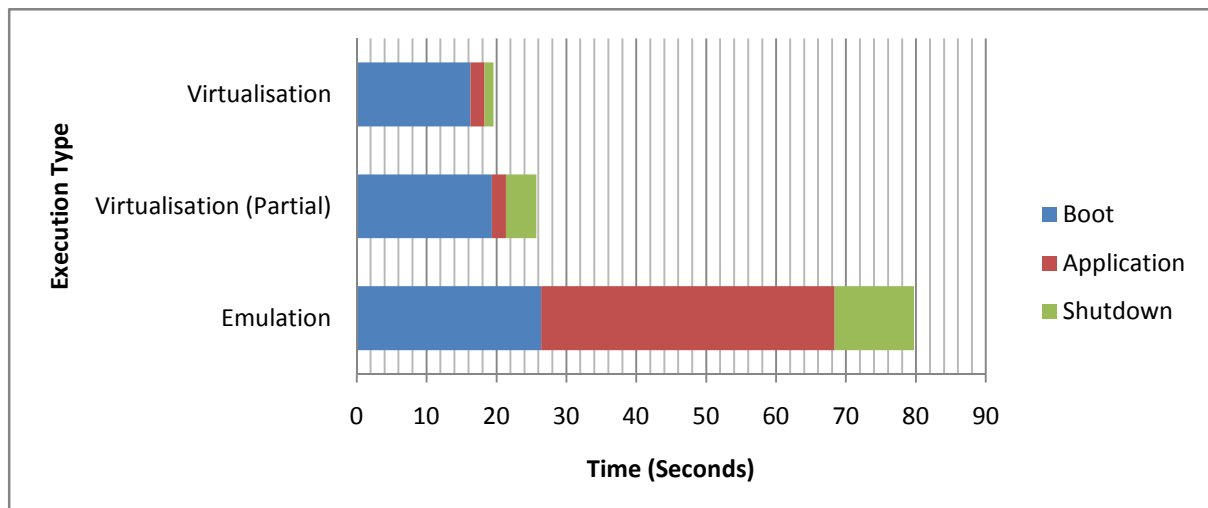


Figure 6.10: Approximate Virtual Machine Work Unit running times

Figure 6.10 shows a graph of the approximate times for starting the Virtual Machine Work Unit. This was run on the workstation using the Mandelbrot grid application which is discussed as an example in Section 6.1.1. The times are approximated and may vary depending on the current state of the host operating system. This graph shows that the overhead in the starting up and shutdown of the guest operating system. The boot time above includes the 15 second delay in the boot loader (GRUB) before the guest operating system is started.

These results are dependent on the setup of the execution environment. However there are a number of ways to minimise the overhead times.

- Ensure the boot loader automatically selects the guest operating system and launches it without delay for a boot selection menu.

- Remove unnecessary services that start-up within the guest operating system.
- Using check pointing to skip the loading process and start the virtual machine just before the grid application is executed.

At this stage the base images used in testing have not utilised these improvements, however it is important when launching grid applications on to the grid using Virtual Machine Work Units that these provisions are provided with pre-configured environments.

Another important metric is the size of Virtual Machine Work Units. This is highly dependent on the base image used, and the size of the grid application. Other factors include the amount of empty space contained on each of these images as well as the format of the virtual machine disk (Raw, VMDK, etc).

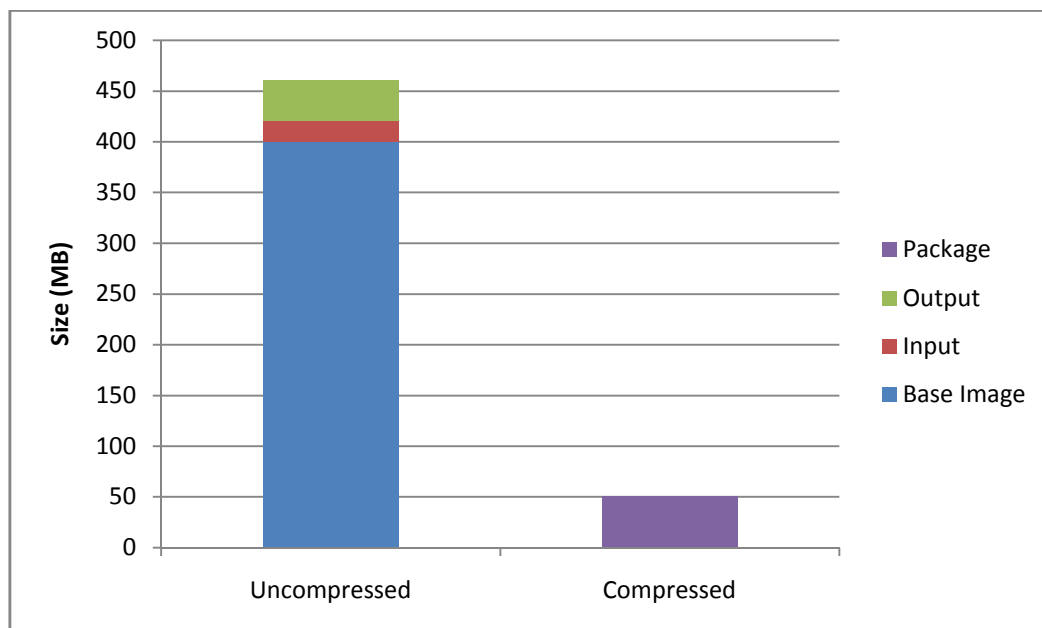


Figure 6.11: Size metrics of Virtual Machine Work Unit components

Figure 6.11 shows the size of Mandelbrot Virtual Machine Work Unit. The base image makes up the majority of the entire package and accounts for a lot of the overhead in transferring Virtual Machine Work Units across the grid. Compressing the entire package dramatically decreases the size of the Virtual Machine Work Unit; this can be attributed to the compression removing the empty space that exists on each virtual machine disk.

6.2.2 Performance of Virtualisation used by Virtual Machine Work Units

The performance of virtualisation is critical to the uptake of virtualisation in grid computing and considerable research has been conducted on high-performance computing using virtual machines as discussed in Section 3.4.1.

However it is worthwhile looking at some issues with deploying Virtual Machine Work Units on a wide range of heterogeneous architectures. In the testing of Virtual Machine Work Units QEMU was used. QEMU is a virtual machine emulator which with driver support can provide virtualisation. However virtualisation can only be achieved when the underlying grid resource architecture is equivalent to the virtual machine architecture. Without this

support, the performance of emulation can be extremely slow. For external information on the performance of QEMU see [81].

QEMU provides three different modes: emulation (-no-kqemu), partial virtualisation of virtual machine not privilege instructions (-kqemu), and virtualisation of virtual machine’s privileged and non-privileged instructions (-kernel-kqemu).

The performance test that was conducted was comparing the three different modes that QEMU provides. This was run on the workstation using the Mandelbrot grid application which is discussed as an example in Section 6.1.1.

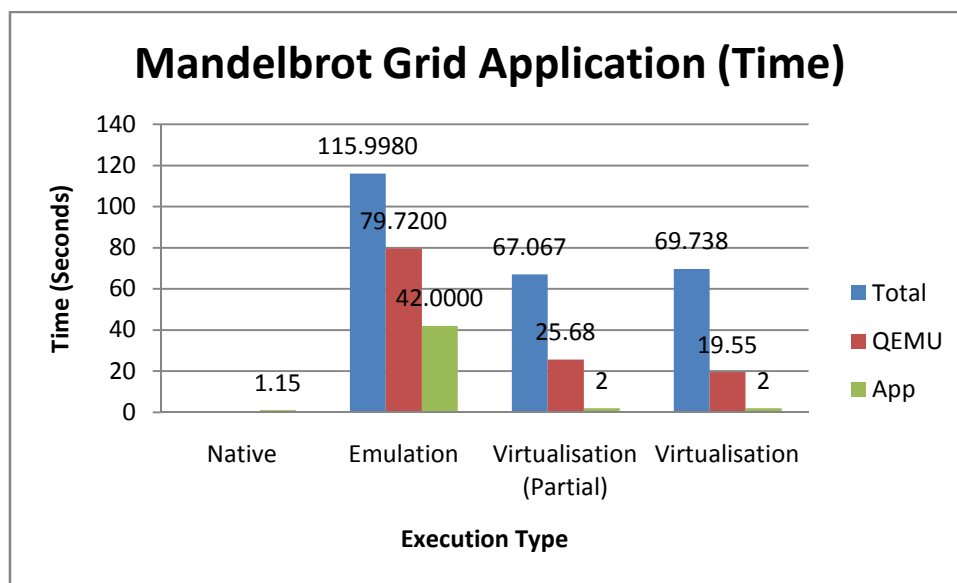


Figure 6.12: Mandelbrot grid application performance comparisons

Figure 6.12 shows the execution times of the Mandelbrot Virtual Machine Work Unit using the parameters “Mandelbrot.out -2.0 1.0 -1.0 1.0 1600 800 100 0 0 1600 800”. Total time represents the entire process of launching the Virtual Machine Work Unit including the decompressing, un-packaging, creation of the ISO image, launching the virtual machine with QEMU, and extracting the output image. This was run on the workstation so that the various modes could be tested. QEMU time represents the amount of time to launch the virtual machine till it terminates, and the application time represents the total time of execution. The application time may be distorted due to timing inaccuracies within the virtual machine as well as the measurements used.

The graph shows us that virtualisation application performance is very close to the native application performance excluding the overhead of the launching utility and virtual machine load times. Emulation however decreases the performance significantly.

6.3 Virtual Machine Work Unit Feasibility

Section 6.1 provides a demonstration on the useability of Virtual Machine Work Units on two different grid applications. Virtual Machine Work Units can be easily created by the e-scientist by simply passing their grid application and its architectural and operating system requirements. The execution of Virtual Machine Work Units on existing infrastructure was also shown utilising both the Sun Grid Engine and Nimrod and demonstrated that even

though the grid application is encapsulated within a Virtual Machine Work Unit it can be launched in the same manner as traditional grid applications.

Virtualisation has often been questioned with use in high performance and much research has taken place on its viability in grid computing as discussed in Section 3.4.1 and in similar projects discussed in Section 3.4.3. Supporting Virtual Machine Work Units on existing grid infrastructure requires some considerations with regards to performance and the design of environments within Virtual Machine Work Units. Section 6.2 highlights the optimisations that can be used to reduce the overhead in Virtual Machine Work Units and also discussed the implications of virtualisation support of grid resources used by Virtual Machine Work Units.

Performance costs can be mitigated with the slight modification of existing grid infrastructure by providing virtualisation capabilities. In this implementation, the use of the KQEMU driver on grid resources reduces the cost of performance of executing Virtual Machine Work Units over QEMU. In some situations the performance of emulation can be tolerable for experiments that are not computationally expensive when run in parallel and when the computational resources are freely available.

Even with the overhead of Virtual Machine Work Units, the ability for the e-scientist to develop and deploy grid applications in their own terms outweighs the performance costs.

Chapter 7 Conclusion and Future Work

Grid computing offers significant promise as the next generation platform which will drive large-scale e-science. However, e-scientists are faced with challenging problems when developing and deploying grid applications. Virtual machines reduce the effort required for developing grid applications by abstracting the resource characteristics and allow e-scientists to define their own run-time environment. However utilising virtual machines for grid computing poses problems for e-scientists as the configuration of such environments can be complex.

The challenges faced in the development and deployment of grid applications and the advantages of platform virtual machines were the motivation behind this thesis and the design of Virtual Machine Work Units. In this section, we review the proposed objectives for Virtual Machine Work Units as discussed in Section 4.1, and also discuss future work needed for the adoption of Virtual Machine Work Units.

7.1 Review

This thesis has presented the design of a flexible grid virtual machine architecture and a framework for the dynamic generation and execution of virtual machines across existing grid infrastructure.

Capitalising on the advantages of platform virtual machines, Virtual Machine Work Units provide the e-scientist with the ability to encapsulate their grid application within an execution environment created or selected to meet the requirements of the grid application. Using virtual machines alleviate the issues of traditional grid application development and deployment that constrain the e-scientist to the provisions of the grid resources both technically and organisationally.

The proposed Virtual Machine Work Unit architecture (Section Chapter 4) was designed to allow the dynamic creation and execution of Virtual Machine Work Units across existing grid infrastructure. The design of Virtual Machine Work Units promotes reuse and allows the Virtual Machine Work Unit and created environments to be utilised by different experiments and grid applications respectively. Grid applications can be packaged into virtual machines of varying architectures different to the underlying architecture of the grid; however providing the freedom of architectural requirements leads to performance costs when the architecture of the virtual machine differs to the architecture of the grid resource.

Supporting utilities (Section Chapter 5) were implemented that include the necessary functionality to generate and execute Virtual Machine Work Units. This includes a Virtual Machine Work Unit packaging utility, virtual machine disk database, Virtual Machine Work Unit launching utility, and Virtual Machine Work Unit framework for abstracting virtual machine disks and virtual machine monitors.

The packaging utility provides the ability for e-scientists to easily encapsulate their grid application in an execution environment that meets its requirements. Grid applications are copied in to the virtual machine and provide the e-scientist with the ability to define the interaction of the grid application within the virtual machine. This includes launching other

scripts to pre-configure the environment and grid application data, and post-process the results of the grid application execution from within the virtual machine.

A virtual machine disk database that contains base images, with pre-configured guest operating systems, provides the e-scientist with the ability to use tested environments for their grid application. This removes the complexity required in setting up their own environments such as installing the Virtual Machine Work Unit daemon that was implemented to support the automated execution of grid applications.

Executing Virtual Machine Work Units on existing grid infrastructure is supported through the implementation of a launching utility. The launching utility provides the necessary mechanisms required for executing Virtual Machine Work Units and allows the e-scientist to pass parameters and files to the grid application within each instance of the Virtual Machine Work Unit. This allowed the same Virtual Machine Work Unit to be executed across multiple grid resources with different outputs. Parameters and files are passed by generating an ISO image and copying supporting data in to the image which from the perspective of the grid application is a CD-ROM. Virtual machine monitor support is currently limited to QEMU; however support for other virtual machine monitors was designed in to the launcher utility and allows the specification of a launching command. The output of the grid application would then be extracted from the Virtual Machine Work Unit for the e-scientist to process.

Supporting both utilities, a Virtual Machine Work Unit framework was created to abstract the implementation details of virtual machine disks and virtual machine monitors. This framework can easily be extended to support other virtual machine disk formats and virtual machine monitors.

The feasibility of Virtual Machine Work Units was analysed looking at both useability and performance (Section Chapter 6). Examples of using Virtual Machine Work Units for experiments demonstrated the effort required by e-scientists to utilise Virtual Machine Work Units on existing grid infrastructure, and how these Virtual Machine Work Units were initially created to meet the requirements of the grid application.

The performance metrics in generating and executed Virtual Machine Work Units were briefly analysed providing some insight to the optimisations that can occur in improving the performance of Virtual Machine Work Units across existing grid infrastructure. Grid infrastructure without supporting virtualisation mechanisms does prohibit the effective usage of Virtual Machine Work Units as the cost of emulation places to a performance barrier on high-performance computing. However, with slight modification virtualisation can be utilised and recent publications have shown that virtualisation is able to achieve performance results needed in high-performance computing.

Virtual Machine Work Units allow the easy encapsulation of grid applications in to virtual machines providing the e-scientist with a platform to develop applications on their terms and not based on the technical and organisational constraints of grid resources. Virtual Machines Work Units can be successfully applied to existing grid infrastructure and with support for virtualisation can be effectively utilised by computationally intensive grid applications.

7.2 Future Work

Expanding the Virtual Machine Work Unit framework to support more virtual machine disk formats and virtual machine monitors is critical to the value of using Virtual Machine Work Units on existing grid infrastructure and utilisation of the packaging and launching utilities.

The pre-configured environments implemented in this example were all Unix-based systems. The packager was designed to support a wide range of platforms; however the packager requires that the pre-configured environments supply a launching script template that can be used when generating the virtual machine auto-run script. Expanding the default pre-configured environments needs to be tested for non Unix-based systems such as Microsoft Windows. Pre-configured environments also need to be optimised to reduce the loading and shut down times within the virtual machine, and provide support for check-pointing when supporting infrastructure is present. The guest operating systems also need to be scaled down to support the bare minimum software configuration needed in the execution of the virtual machine. All these steps are necessary to reduce wasted computational resources.

The virtual appliance design of the Virtual Machine Work Unit provides the necessary structure for launching Virtual Machine Work Units independent of the virtual machine monitor. Support for the OVF standard was designed into the Virtual Machine Work Unit but has not been implemented for providing the virtual machine monitor independent functionality provided by virtual appliances.

The Virtual Machine Work Unit daemon present in the pre-configure environments provides the basic functionality for combining the Virtual Machine Work Unit components and launching scripts passed in for executing the grid application. Due to the usage of such a daemon, expanding its functionality can aid in the testing and monitoring of the environment within the virtual machine. This information can then be provided to the e-scientist during the experiment.

Virtual Machine Work Units are designed to be loosely coupled from the underlying grid resource. However further analysis of utilising different pre-configured environments needs to be tested in real-world scenarios. This includes testing and quantifying the performance costs when running Virtual Machine Work Units that support different architectural and operating system than the underlying grid resources being utilised.

Virtual Machine Work Units are designed to support data and computational independent grid applications. This limitation does not allow the execution of parallel grid applications that communicate with each other during execution, such as applications built on the MPI framework[23]. Virtual machines can be configured with network connectivity and are able to support the execution of parallel applications. Virtual Machine Work Units could be expanded to support parallel grid applications with modifications to the grid infrastructure. Supporting such architecture is complex and requires supporting mechanisms to be implemented on existing grid infrastructure for the creation of a virtual network to facilitate communication; some virtual machine grid architecture have utilised this in supporting parallel applications[43, 46, 49].

References

1. Foster, I. and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. 1999, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
2. Goscinski, W. and D. Abramson. *Distributed Ant: A System to Support Application Deployment in the Grid*. in *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*. 2004: IEEE Computer Society.
3. Goscinski, W. and D. Abramson. *Application Deployment over Heterogeneous Grids using Distributed Ant*. in *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*. 2005: IEEE Computer Society.
4. Figueiredo, R.J., P.A. Dinda, and J. Fortes. *A Case For Grid Computing On Virtual Machines*. in *23rd IEEE International Conference on Distributed Computing Systems (ICDCS'03)*. 2003.
5. Jankowski, N.W., *Exploring e-Science: An Introduction*. Computer-Mediated Communication, 2007. **12**(2).
6. Nentwich, M., *Cyberscience - Research in the Age of the Internet*. 2003, Vienna: Austrian Academy of Sciences Press.
7. Wikipedia.org. *e-Science*. [cited; Available from: <http://en.wikipedia.org/wiki/E-Science>].
8. Kuck, D.J., *High Performance Computing: Challenges for Future Systems*. 1996, New York: Oxford University Press.
9. Sterling, T.L., *Beowulf Cluster Computing with Linux*. 2002: MIT Press. 496.
10. 500, T., *Top 500 Supercomputer Sites*. 2007.
11. Sterling, T.L., et al. *Beowulf: A Parallel Workstation for Scientific Computation*. in *Proceedings of the 24th International Conference on Parallel Processing*. 1995. Oconomowoc, WI.
12. Castagnera, K., et al., *Clustered Workstations and their Potential Role as High Speed Compute Processors*. 1994, NASA Ames Research Center: Moffett Field, CA.
13. Gentzsch, W., *Sun Grid Engine: Towards Creating a Compute Power Grid*, in *International Symposium on Cluster Computing and the Grid*. 2001: Brisbane, Australia. p. 35-36.
14. Foster, I., *The Grid: A new infrastructure for the 21st century science*, in *Physics Today*. 2002, American Institute of Physics.
15. Berstis, V., *Fundamentals of Grid Computing*. IBM Redbooks Paper, 2002.
16. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. Lecture Notes in Computer Science, 2001. **2150**.
17. Anderson, D.P., et al., *SETI@home: An Experiment in Public-Resource Computing*. Communications of the ACM, 2002. **45**(11): p. 56-61.
18. Stanford. *Genome@home*. 2003 [cited; Available from: <http://genomeathome.stanford.edu/>].
19. Jacob, B., et al., *Introduction to Grid Computing*. 2005, IBM Redbooks.
20. Abbas, A., *Grid Computing Technology - An Overview*, in *Grid Computing: A Practical Guide to Technology and Applications*, D. Pallai, Editor. 2003, Charles River Media Inc.: Hingham. p. 43-74.
21. Foster, I. and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*. The International Journal of Supercomputer Applications and High Performance Computing, 1997. **11**(2): p. 115-128.

22. Globus. *Towards Open Grid Services Architecture*. 2008 [cited; Available from: <http://www.globus.org/ogsa/>].
23. Wikipedia.org. *Message Passing Interface (MPI)*. 2008 [cited; Available from: http://en.wikipedia.org/wiki/Message_Passing_Interface].
24. Abramson, D., et al., *Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations*, in *4th IEEE Symposium on High Performance Distributed Computing*. 1995: Virginia.
25. Buyya, R., D. Abramson, and J. Giddy, *Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid*, in *HPC Asia*. 2000. p. 283-289.
26. Abramson, D., J. Giddy, and L. Kotler, *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?*, in *International Parallel and Distributed Processing Symposium (IPDPS)*. 2000: Cancun, Mexico. p. 520-528.
27. Abramson, D., *Applications Development for the Computational Grid*. Lecture Notes in Computer Science, 2006. **3841**: p. 1 – 12.
28. Sunderam, V.S., *PVM: A Framework for Parallel Distributed Computing*. Concurrency: Practice and Experience, 1990. **2**(4): p. 315-339.
29. Goscinski, W. and D. Abramson, *An Infrastructure for the Deployment of e-Science Applications*.
30. Berman, F., et al., *The GrADS Project: Software Support for High-Level Grid Application Development*. The International Journal of High Performance Computing Applications, 2001. **15**(4): p. 327-344.
31. Smith, J.E. and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*. 2005, San Francisco, CA, USA: Elsevier Inc.
32. Goldbery, R.P., *Survey of Virtual Machine Research*. IEEE Computer, 1974. **7**(6): p. 34-45.
33. Ferrie, P. *Attacks on Virtual Machine Emulators*. in *Symantec Technology Exchange*. 2007.
34. VMWare. *VMWare Virtualization*. 2008 [cited 2008 June]; Available from: <http://www.vmware.com/virtualization/>.
35. Barham, P., et al., *Xen and the art of virtualization*. SIGOPS Oper. Syst. Rev., 2003. **37**(5): p. 164-177.
36. Microsoft. *Virtualization Solutions: Windows Server*. 2008 [cited 2008 June]; Available from: <http://www.microsoft.com/virtualization/solution-product-ws.msp.x>.
37. Bellard, F. *QEMU*. 2008 [cited; Available from: <http://bellard.org/qemu/>].
38. VMWare. *Virtual Machine Disk Format*. 2008 [cited 2008 June]; Available from: <http://www.vmware.com/interfaces/vmdk.html>.
39. VMWare, *VMware Virtual Disks - Virtual Disk Format 1.1*. 2007.
40. QEMU. *Disk Images*. 2008 [cited; Available from: <http://bellard.org/qemu/qemu-doc.html#SEC15>].
41. VMWare. *Virtual Disk Manager*. 2008 [cited 2008 June]; Available from: <http://www.vmware.com/support/developer/vddk/VirtualDiskManager.pdf>.
42. Tannenbaum, T., et al., *Condor: a distributed job scheduler*. Beowulf cluster computing with Linux, 2002: p. 307-350.
43. Adabala, S., et al., *From virtualized resources to virtual computing grids: the In-VIGO system*. Future Gener. Comput. Syst., 2005. **21**(6): p. 896-909.

44. Goscinski, W. and D. Abramson, *Motor: A Virtual Machine for High Performance Computing*. 2006: p. 171-182.
45. Haase, J., F. Eschmann, and K. Waldschmidt. *The Self Distributing Virtual Machine (SDVM) - Making Computing Clusters Heal Themselves*. in *Proceedings of the 23rd IASTED Internation Multi-Conference Parallel and Distributed Computing and Networks*. 2005. Innsbruck, Austria.
46. Joomla! *Grid Appliances*. 2008 [cited; Available from: <http://www.grid-appliance.org/>].
47. Keahey, K., K. Doering, and I. Foster. *From Sandbox to Playground: Dynamic Virtual Environments in the Grid*. in *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*. 2004: IEEE Computer Society.
48. Keahey, K., et al., *Virtual workspaces: Achieving quality of service and quality of life in the Grid*. *Sci. Program.*, 2005. **13**(4): p. 265-275.
49. Keahey, K., et al. *Virtual Workspaces in the Grid*. in *Europar*. 2005. Lisbon, Portugal.
50. Krsul, I., et al. *VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing*. in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. 2004: IEEE Computer Society.
51. Santhanam, S., et al. *Deploying virtual machines as sandboxes for the grid*. in *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*. 2005. San Francisco, CA: USENIX Association.
52. Wolinsky, D.I., et al. *On the Design of Virtual Machine Sandboxes for Distributed Computing in Wide-area Overlays of Virtual Workstations*. in *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. 2006: IEEE Computer Society.
53. Cherkasova, L., et al. *Optimizing grid site manager performance with virtual machines*. in *WORLDS'06: Proceedings of the 3rd conference on USENIX Workshop on Real, Large Distributed Systems*. 2006. Seattle, WA: USENIX Association.
54. Chase, J.S., et al. *Dynamic Virtual Clusters in a Grid Site Manager*. in *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. 2003. Washington, DC, USA: IEEE Computer Society.
55. Ruth, P., et al., *Enabling Autonomic Adaptation of Virtual Computational Environments in a Shared Distributed Infrastructure*.
56. Childs, S., et al. *A Single-Computer Grid Gateway Using Virtual Machines*. in *AINA '05: Proceedings of the 19th International Conference on Advanced Information Networking and Applications*. 2005. Washington, DC, USA: IEEE Computer Society.
57. Bannon, D., et al. *Experiences with a Grid Gateway Architecture Using Virtual Machines*. in *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International Workshop on*. 2006.
58. Macdonell, C. and P. Lu. *Pragmatics of Virtual Machines for High-Performance Computing: A Quantitative Study of Basic Overheads*. in *2007 High Performance Computing & Simulation Conference (HPCS'07)*. 2007. Prague, Czech Republic.
59. Menon, A., et al. *Diagnosing Performance Overheads in the Xen Virtual Machine Environment*. in *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*. 2005. Chicago, IL, USA: ACM.
60. Adams, K. and O. Agesen. *A comparison of software and hardware techniques for x86 virtualization*. in *ASPLOS-XII: Proceedings of the 12th international conference on*

- Architectural support for programming languages and operating systems*. 2006. San Jose, California, USA: ACM.
61. Sapuntzakis, C., et al. *Virtual Appliances for Deploying and Maintaining Software*. in *LISA '03: Proceedings of the 17th USENIX conference on System administration*. 2003. San Diego, CA: USENIX Association.
 62. Project, I. *IPOP*. 2008 [cited; Available from: <http://www.ipop-project.org/>].
 63. Wikipedia.org. *Virtual appliance*. [cited; Available from: http://en.wikipedia.org/wiki/Virtual_appliance].
 64. VMWare. *Virtual Appliance Frequently Asked Questions*. [cited; Available from: <http://www.vmware.com/appliances/learn/faq.html>].
 65. Krishnamurti, S. *Get Juiced!* 2007 [cited 2008 17 October]; Available from: <http://blogs.vmware.com/console/2007/07/get-juiced.html>.
 66. FastScale. *FastScale Composer Suite*. 2008 [cited; Available from: <http://www.fastscale.com/technology/composer.shtml>].
 67. VMWare. *Open Virtualization Format*. 2008 [cited; Available from: <http://www.vmware.com/appliances/learn/ovf.html>].
 68. Bellard, F. *QEMU, a Fast and Portable Dynamic Translator*. in *USENIX 2005 Annual Technical Conference, FREENIX 2005*.
 69. *QEMU Accelerator Technical Documentation*. [cited; Available from: <http://bellard.org/qemu/kgemu-tech.html>].
 70. Wikipedia.org. *Syslog*. 2008 [cited 2008 26 October]; Available from: <http://en.wikipedia.org/wiki/Syslog>.
 71. *losetup - Linux Manual Page*. [cited; Available from: http://linuxcommand.org/man_pages/losetup8.html].
 72. *fdisk(8) - Linux Manual Page*. [cited; Available from: <http://linux.die.net/man/8/fdisk>].
 73. *mkfs(8) - Linux Manual Page*. [cited; Available from: <http://linux.die.net/man/8/mkfs>].
 74. *mount(8) - Linux Manual Page*. [cited; Available from: <http://linux.die.net/man/8/mount>].
 75. *umount(2) - Linux Man Page*. [cited; Available from: <http://linux.die.net/man/2/umount>].
 76. *mkisofs(8) - Linux Manual Page*. [cited; Available from: http://linuxcommand.org/man_pages/mkisofs8.html].
 77. *growisofs(1m) - Linux Manual Page*. [cited; Available from: http://linuxcommand.org/man_pages/growisofs1.html].
 78. Laboratory, M.e.a.G.E. *Infrastructure*. 2008 [cited; Available from: <http://messagelab.monash.edu.au/infrastructure>].
 79. Wikipedia.org. *Mandelbrot Set*. 2008 [cited; Available from: http://en.wikipedia.org/wiki/Mandelbrot_set].
 80. McCoy, A.J., et al., *Phaser crystallographic software*. *J. Appl. Cryst*, 2007. **40**: p. 658-674.
 81. *Finally user-friendly virtualization for Linux*. 2006 [cited; Available from: <http://www.linuxinsight.com/finally-user-friendly-virtualization-for-linux.html>].

Appendix A – Virtual Machine Work Unit Packager Configuration Options

The following table documents all the configuration settings available to the e-scientist in using the Virtual Machine Work Unit Packager. These can either be passed in as command-line arguments or using the configuration file.

Option	Description
app-architecture	Specifies the required architecture of the grid application. Options are dependent on the virtual machine disk database; however in the test implementation an example includes “x86” architecture. This value is checked against the database. Represented as a string.
app-buffer	Represents the amount of buffer space required by the grid application. This increases the size of the virtual machine disk to allow the required space for any temporary files that are generated by the grid application. Represented as an integer that represents the size in bytes.
app-cdrom-dir	Specifies the directory of where the CD-ROM is mounted to within the virtual machine. In UNIX-based systems this defaults to “/cdrom/”, however it may be necessary for the e-scientist to change this mounting point. Represented as a file location.
app-dir	Specifies the directory of where the application image virtual machine disk should be mounted within the virtual machine. This is prescribed by the pre-configured base image and the setup of the virtual machine disk work unit daemon within the guest operating system. Represented as a file location.
app-import-dir	Specifies the directory of the files and sub-directories that contain the grid application. Any contents within this directory will be transferred to the application image of the Virtual Machine Work Unit. The directory specified will be the root level of the application image. Represented as a file location.
app-launch	Specifies the launching command for the grid application. If the grid application output is directed to standard out then this launching command should include stdout redirection that points to the output directory specified by “app-output-dir”. Redirecting stderr may also be useful in debugging any grid application errors that occur during the instantiation of the Virtual Machine Work Unit. If the application requires parameters at run-time than this can be accomplished by appending “`cat /cdrom/param`” to the launch command. The application executable should be the absolute path, e.g. /bin/sleep. Represented as a string.
app-os	Specifies the required operating system of the grid application. Options are dependent on the virtual machine disk database; however in the test implementation an example includes “linux2.6” based operating system. This value is checked against the database. Represented as a string.
app-output-dir	Specifies the directory of where the output image virtual machine disk is mounted to within the virtual machine. In UNIX-based systems this defaults to “/output/”, however it may be required for the e-scientist to change this mounting point. Represented as a file location.
app-output-size	Represents the size of the output image virtual machine disk. This is set to estimated size of results and any other generated files produced by the grid application. Represented as an integer that represents the size in bytes.
app-working-dir	Specifies the working directory when the application is being executed within the virtual machine. By default this is the output directory. Represented as a file location.
base-image	Specifies the file location of the base image to be used by the Virtual Machine

	Work Unit. Provides the ability for e-scientists to use their own pre-configured execution environments. Represented as a file location.
base-image-autosource	Specifies if a pre-configured base image should be sourced to meet the requirements of the grid application. This removes the effort required by the e-scientist in creating an execution environment. Represented as a Boolean integer value (0 or 1).
base-image-autosource-interaction	Specifies if when sourcing a pre-configured base image, the e-scientist should choose from a list of possible execution environments. If set to false, the first possible match is selected. Used in conjunction with the “base-image-autosource” option. Represented as a Boolean integer value (0 or 1).
base-image-cdrom-mount-device	Specifies the device that represents the CD-ROM within the virtual machine. This allows the CD-ROM image created by the launcher to be mounted at instantiation. Represented as a string.
base-image-filesystem	Specifies the file system format of the base image. This ensures that the application image and output image can be installed and accessed within the virtual machine. Represented as a string.
base-image-input-mount-device	Specifies the device that represents the application image virtual machine disk. This allows the application image to be mounted. This is dependent on the execution environment that is preconfigured. Represented as a string.
base-image-output-mount-device	Specifies the device that represents the output image virtual machine disk within the virtual machine. This allows the output image to be mounted at instantiation. Represented as a string.
cdrom-as-hard-disk	Specifies if the virtual machine monitor should use the ISO image generated as a hard-disk rather as a CD-ROM. Provides support for guest operating systems that have trouble mounting ISO9660 CD-ROMs. This is only necessary if the e-scientist wants to edit the execution environment before packaging the Virtual Machine Work Unit. Represented as a string.
compress	Specifies if the Virtual Machine Work Unit generated should be compressed. Represented as a Boolean integer value (0 or 1).
config	Argument only. Specifies the configuration file to be used by the packager and the options for the creation of the Virtual Machine Work Unit. Any configuration file options will be overwritten by corresponding command-line arguments. Represented as a file location.
current-directory	Specifies the current directory to be used when running the packaging utility. Represented as a file location.
edit-script	Specifies if the packager should launch a text editor to modify the virtual machine auto-run script before packaging the Virtual Machine Work Unit. Allows the e-scientist to provide any more commands that are needed by the grid application. Represented as a string.
edit-vm	Specifies if the e-scientist wants to edit the execution environment before the Virtual Machine Work Unit is packaged. Represented as a Boolean integer value (0 or 1).
input-import-dir	Specifies the directory that contains any data that needs to be bundled with the Virtual Machine Work Unit. Any files or sub-directories within this directory are copied to the application image and contained in a directory called “data”. Represented as a file location.
instance-post-script-cmd	Specifies a command to be inserted and executed after the launching of the grid application. This can point to execute a script contained on the cd-rom passed in at instantiation or a script contained within the application image. Represented as a string.

instance-script-cmd	Specifies a command to be inserted and executed before the launching of the grid application. This can point to execute a script contained on the cd-rom passed in at instantiation or a script contained within the application image. Represented as a string.
package-name	Specifies the name of the Virtual Machine Work Unit outputted by the packager. Represented as a string.
script-name	Specifies the name of the virtual machine auto-run script generated by the packager and launched within the virtual machine. Represented as a string.
script-template	Specifies the location of the template file used to create the virtual machine auto-run script. This is usually dependent on the base image used. If the base image is auto sourced then this information is retrieved from the virtual machine disk database. Using this option will overwrite the template script used. Represented as a file location. Represented as a file location.
text-editor	Specifies the command to launch the text editor used to edit the virtual machine auto-run script. By default it uses ViM. Represented as a string.
vmddb	Specifies the location or hostname of the virtual machine disk database. Represented as a file location or string.
vmm-architecture	Specifies the architecture of the virtual machine to be launched on the virtual machine monitor. This is only necessary if the e-scientist wants to edit the execution environment before packaging the Virtual Machine Work Unit. Represented as a string.
vmm-directory	Specifies the directory that contains the virtual machine monitor. This is only necessary if the e-scientist wants to edit the execution environment before packaging the Virtual Machine Work Unit. Represented as a string.
vmm-executable	Specifies the executable name of the virtual machine monitor. This is only necessary if the e-scientist wants to edit the execution environment before packaging the Virtual Machine Work Unit. Represented as a string.
vmm-local	Specifies if the virtual machine monitor is in a local directory. This is only necessary if the e-scientist wants to edit the execution environment before packaging the Virtual Machine Work Unit. Represented as a string.
vmm-type	Specifies the virtual machine monitor to be used to launch the Virtual Machine Work Unit. This is only necessary if the e-scientist wants to edit the execution environment before packaging the Virtual Machine Work Unit. Represented as a string.

Appendix B – Virtual Machine Auto-Run Script Template Token Substitutions

The following table documents all the tokens in script templates that are replaced by the packager utility when creating the Virtual Machine Work Unit Auto-Run Script.

Token	Description
###APPLICATION_DIRECTORY###	Specifies the directory within the virtual machine that contains the application
###CDROM_DIRECTORY###	Specifies the directory within the virtual machine that contains the files of the CD-ROM.
###CDROM_FILESYSTEM_FORMAT###	Specifies the file system format of the CD-ROM image.
###CDROM_MOUNT_DEVICE###	Specifies the device that represents the CD-ROM within the virtual machine.
###INPUT_DIRECTORY###	Specifies the directory within the virtual machine that contains the applications data.
###INPUT_MOUNT_DEVICE###	Specifies the device that represents the application image virtual machine disk within the virtual machine.
###INSTANCE_COMMAND###	Specifies the command to be executed before the application is launched within the virtual machine.
###INSTANCE_POST_COMMAND###	Specifies the command to be executed after the application is launched within the virtual machine.
###LAUNCHING_COMMAND###	Specifies the launching command to execute the grid application within the virtual machine.
###OUTPUT_DIRECTORY###	Specifies the directory within the virtual machine that the application can output to.
###OUTPUT_FILESYSTEM_FORMAT###	Specifies the file system format of the output image virtual machine disk.
###OUTPUT_MOUNT_DEVICE###	Specifies the device that represents the output image virtual machine disk within the virtual machine.
###WORKING_DIRECTORY###	Specifies the working directory of the grid application.

Appendix C – Virtual Machine Work Unit Packager Utility Dependencies

The Virtual Machine Work Unit Packager uses existing programs and libraries in the creation of Virtual Machine Work Units. The prerequisites and dependencies of the packager are listed below.

The following Perl Modules and Packages were used by the packaging utility and include:

- GetOpt::Long
- Config::Fast
- File::Temp
- File::Basename
- File::Spec::Functions
- Cwd
- DBI

The following programs were used by the packaging utility and include:

- chmod
- tar
- gzip
- qemu-img
- mount
- umount
- losetup
- mkfs
- fdisk

Appendix D – Virtual Machine Work Unit Launcher Configuration Options

The following table documents all the configuration settings available to the e-scientist in using the Virtual Machine Work Unit Launcher. These can either be passed in as command-line arguments or using the configuration file.

Option	Description
base-image	Specifies the location of the base image. This option enforces the launcher to ignore the base image in the Virtual Machine Work Unit and use the specified one. Represented as a file location.
cdrom-as-hard-disk	Specifies if the ISO image should be mounted to the virtual machines hard-drive device rather than CD-ROM device. Needed in case guest operating system doesn't support emulated virtual CD-ROM devices. Represented as a Boolean integer value (0 or 1).
cdrom-image	Specifies the location of the ISO image. This option enforces the launcher to ignore the ISO image in the Virtual Machine Work Unit and use the specified one. Represented as a file location.
compressed	Specifies if the Virtual Machine Work Unit being launched is compressed. Represented as a Boolean integer value (0 or 1).
config	Argument only. Specifies the configuration file to be used by the launcher and the options for launching the Virtual Machine Work Unit. Any configuration file options will be overwritten by corresponding command-line arguments. Represented as a file location.
current-directory	Specifies the current directory to be utilised by the launcher. Represented as a file location.
files	A list that specifies the files to be copied to the generated ISO image. Format is <file1>,<file2>,...,<file n>. Represented as a string.
files-destination	A list that is used in conjunction with the files option. Specifies the corresponding location within the generated ISO image. If not present, it assumes the destination is the root of the CD-ROM.
input-image	Specifies the location of the input image. This option enforces the launcher to ignore the input image in the Virtual Machine Work Unit and use the specified one. Represented as a file location.
instance-post-script	Specifies the script to be launched within the virtual machine after the grid application is executed. Represented as a file location.
instance-script	Specifies the script to be launched within the virtual machine before the grid application is executed. Represented as a file location.
output-directory	Specifies the directory the output image should be copied to. Represented as a file location.
output-filename	Specifies the name of the output image. Represented as a string.
output-image	Specifies the location of the output image. This option enforces the launcher to ignore the output image in the Virtual Machine Work Unit and use the specified one. Represented as a file location.
overwrite	Specifies if the launcher should overwrite the Virtual Machine Work Unit package with the changes that occur during the execution. Represented as a Boolean integer value (0 or 1).
parameters-filename	Specifies the name of the file that contains the parameters located on the generated ISO image. Represented as a string.
vmm-	Specifies the architecture of the virtual machine to be launched on the virtual

architecture	machine monitor. Represented as a string.
vmm-directory	Specifies the directory that contains the virtual machine monitor. Represented as a string.
vmm-executable	Specifies the executable name of the virtual machine monitor. Represented as a string.
vmm-graphics	Specifies if the virtual machine monitor should enable graphic output. Represented as a Boolean integer value (0 or 1).
vmm-local	Specifies if the virtual machine monitor is in a local directory. Represented as a string.
vmm-memory	Specifies the size of the memory on the virtual machine. Default is 512MB. Represented as an integer.
vmm-no-virtualisation	Specifies if the virtual machine monitor should launch the virtual machine in emulation mode and disable any virtualisation support. Represented as a Boolean integer value (0 or 1).
vmm-type	Specifies the virtual machine monitor to be used to launch the Virtual Machine Work Unit. Represented as a string.
vmwu	Specifies the Virtual Machine Work Unit to be launched. Represented as a file location.

Appendix E – Virtual Machine Work Unit Launcher Utility Dependencies

The Virtual Machine Work Unit Launcher uses existing programs and libraries in the execution of Virtual Machine Work Units. The prerequisites and dependencies of the launcher are listed below.

The following Perl Modules and Packages were used by the launching utility and include:

- GetOpt::Long
- Config::Fast
- File::Temp
- File::Basename
- File::Spec::Functions
- Cwd
- Sys::Hostname

The following programs were used by the packaging utility and include:

- tar
- gzip
- qemu
- growisofs
- mkisofs

Appendix F – Test Result Tables

Results conducted and presented in this thesis are based on the following data.

Table 7.1: Virtual Machine Work Unit start-up, application, and shutdown timings. Values are in seconds

	Emulation	Virtualisation (Partial)	Virtualisation
Total	79.7200	25.68	19.55
Boot	26.3600	19.34	16.275
Application	42.0000	2	2
Shutdown	11.36	4.34	1.275

Table 7.2: Size metrics of the Virtual Machine Work Unit. Values are in megabytes

	Uncompressed	Compressed
Base Image	400	
Input	20.4599609	
Output	40	
Package		51.0429106

Table 7.3: Performance results of Virtual Machine Work Units using the Mandelbrot grid application with different virtual machine execution types. Values are in seconds

	Native	Emulation	Virtualisation (Partial)	Virtualisation
Total		115.9980	67.067	69.738
QEMU		79.7200	25.68	19.55
App	1.15	42.0000	2	2