Dynamically Generating and Orchestrating Virtual Machines on the Grid

by

Timothy H. Ward

Literature Review

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

June, 2008

TABLE OF CONTENTS

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. However, there remains many issues in the adoption of grid computing, and as such virtualisation can be applied to solve these many issues. This project examines how platform virtual machine environments can be automatically and dynamically created as work units and how they could be incorporated into a workflow. This includes a review of the issues related with the development and deployment software on to platform virtual machines, and how platform virtual machines can be deployed, maintained, and orchestrated across the grid, therefore 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.

To greater understand what is required for incorporating virtualisation into grid computing; we will first look at high-performance computing and e-Science. This will look at the evolution of e-Science (Section 2) from its early period starting with cluster computing, its evolution in to grid computing, the introduction of scientific workflows, and finally the use of large-scale e-Science. An understanding of the problems associated with the uptake and use of grid computing will be analysed to gain a greater understanding of what is required by e-Scientists in modern day computing for experimentation on the large-scale e-Science infrastructure. This will then be followed by a more technical look at virtualisation (Section 3), specifically examining virtualisation in high-performance computing, and how it is applied in grid computing. Emerging architectures will be discussed with a concentration on platform virtual machines as work units. Finally this review will examine how platform virtual machine work units can be orchestrated across the grid including the configuration, deployment and execution of such units.

2 E-SCIENCE

E-Science [1], also known as cyber-science, is the use of computing resources in aiding and supporting of complex experimentation. Nentwick[2] is 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 [3].

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 presence such as algorithms that are 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[4] 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.

In response to the needs of high-performance computing, various computing paradigms have been created and include parallel programming techniques. One such paradigm is the message passing interface (MPI)[5]. 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[5]. Support for MPI is implemented in most programming languages through APIs.

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[6] 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[7].

The Beowulf cluster[8] is a prominent example of cluster computing. An implementation of networks of workstations[9] 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 there 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[10].

2.3 GRID COMPUTING

Cluster computing offers opportunities of high-performance computing, however it required that resources were dedicated and homogeneous in nature. However 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[11].

The concept of the computational grid was introduced as an analogy to the power grid of the 20th century[11]. 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"[10]. 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[10].

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 [10]:

- 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[12]. Grid resources can include computational resources, storage resources, network resources, and instrumentation resources[13].

Grid computing middleware is still maturing. However, its use in e-Science is being utilised in many scientific projects. Undoubtedly the most recognised use of grid computing is SETI@home[14] and Genome@home[15] 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.

Using the grid is important for the adoption of the grid. The logical process of interaction between the grid and user is usually as follows[16]:

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[17]. 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[18]. Foster and Kesselman define Globus as a low-level toolkit that provides “basic mechanisms such as communication, authentication, network information, and data access”[18]. 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[18]. 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)[19] in which the models and designs of grid architecture and web service architecture is being combined[17].

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 decades[10]. 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 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 maybe accessed without authorisation. Vice versa, grid applications running may invertible maliciously attack the grid resource. Such problems as these limit the potential of grid computing.

2.4 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[11].

Foster and Kesselman[10] 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[20, 21].

Traditional software life-cycles follow a development, deployment, testing and debugging, and execution and this applicable to the development of grid software[22]. The two major challenges in grid computing are development and deployment of grid applications.

Development of grid software 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[23]. 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 schedules and managed across the grid[22]. Unfortunately, during this execution grid software could fail and such software requires recovery mechanisms.

Deployment of grid software 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 applications deployment and execution. This can pose issues to developers unless the abstraction of such resources is utilised. Grid software with specific requirements can only be deployed to grid resources that support these requirements. These requirements are usually specified by the developer.

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”.

One such grid framework that implements this upper middleware layer is the “Infrastructure for the Deployment of e-Science Applications” (IDEA)[24]. IDEA provides tools for managing deployment across grid heterogeneously through DistAnt[20, 21]. 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[25].

2.5 Scientific Workflows

The complexity of experiments is ever increasing and with the availability of grid computing for high-performance computing experiments will get more complex. As such to reduce the complexity of experiments for e-Science, scientific workflows are used. This is another approach to the creation and deployment of grid software.

Scientific workflows involve the breakdown of an experiment into logical and ordered components that may be responsible for the collecting and processing of data. These components may be dependent or a dependency for other components within the workflow. Yu and Buyya define scientific workflows as the “automation of scientific processes in which tasks are structured based on their control and data dependencies”[26]. Scientific workflows have simplified the process of designing and executing scientific experiments. As such applications can be created that can specify computing resources to be used and how they will be orchestrated[26]. To support this paradigm of development for e-Scientists grid workflow systems have been created and utilised. Such workflow systems provide the ability for e-Scientists to design their workflow and composition, provide workflow scheduling support, facilities for fault tolerance, and finally methods for the movement of data[26].

Kepler[27] is a scientific workflow system that allows users to define a workflow using a high-level workflow design. It integrates this with execution and runtime interaction and provides access to local and remote services and data[27]. Kepler allows execution of jobs on the grid by allowing elements or components to be representative of jobs. These include jobs for certificate-based authentication, grid job submissions, and grid-based data access. Kepler also provides ways of designing database accesses and queries into the workflow design[27]. It also allows the execution of programs based in different languages such as Python, and also provides tools for data processing such as the use of Perl[27].

Scientific workflow systems, such as Kepler, reduce the effort required by e-Scientists to orchestrated and conduct scientific experiments on a grid. However, the execution of e-Scientist specific programs is still dependent on the infrastructure provided by the grid, though the design of such workflow systems allows easy modification for supporting new program execution paradigms.

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. Virtualisation is the portioning 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[28]. 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[29]. 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.

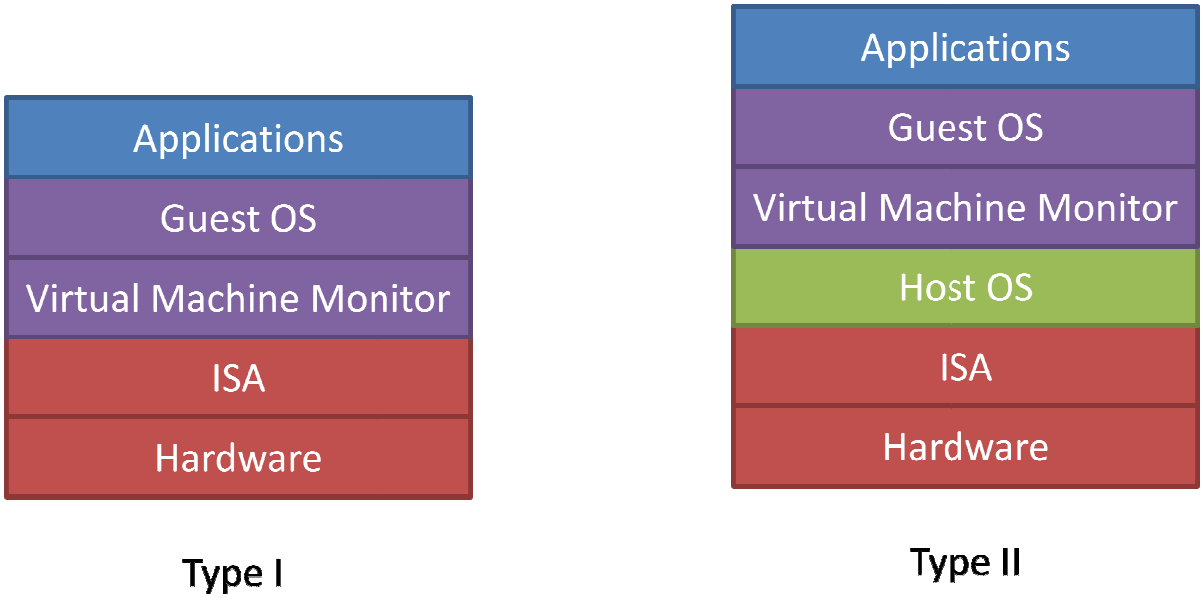

FIGURE 1 - PLATFORM VIRTUAL MACHINE IMPLEMENTATION TYPES

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 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[30].

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.

Finally another important attribute of virtual machines is there 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 and will only discuss a few implementations that are relevant to the project.

QEMU[31] 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; and has been ported to many different architectures[32]. It achieves emulation by using a dynamic translator. QEMU can also be extended to support native instruction execution virtualisation by using a device driver known as KQEMU. QEMU emulator is implemented as an application and as such can be launched as a process within a host operating system without requiring administrator access. QEMU support multiple file formats for storing virtual machine images, these include QCOW, QCOW2, and RAW.

VMWare a leader in commercial virtualisation products also implement a virtual machine monitor[33]. VMWare implements the virtual machine monitor in type I and type II implementations known as VMWare GSX and ESX respectively. There type I virtual machine monitor is based on slimed 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[34] 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[35]. 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.

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)[36].

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[37].

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[38]. 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[39], 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[18, 40]. However, even with such toolkits and standards, implementing infrastructure and developing for grid computing still remains a challenge[22, 24]. In response, virtualisation is being applied as a solution to this problem[41]. Virtual machines have been successfully applied to grid computing, using both application level virtualisation and platform level virtualisation[41-56].

Application virtual machines that make use of .Net/Java virtual machine technologies are already implemented as middleware across different grid implementations[43]. 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[43]. 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[29]. Platform virtual machines provide isolation, legacy-support, administrator privileges, resource control, and environment recovery[41]. 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[57]. 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[45]. The second approach is using existing grid infrastructure and using virtual machines as a work units for the execution of applications [42, 49, 50]. 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[41, 58, 59] as well as there feasibility for high-performance computing[57].

Macdonnell and Lu[57] 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[57]. 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 there 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[57].

Application virtual machines have been developed for high-performance computing purposes. One such example is Motor[43]. Motor takes advantages of virtualisation and modifies an existing virtual machine, the Common Runtime Infrastructure (CLI). 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[43].

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[60], 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[45] 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[61]. 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[41]. Virtual machine images are submitted as jobs rather than the applications and can be incorporated into scientific workflow systems such Kepler. 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 machine work units are often referred to as sandboxes as they allow users to customise the execution environment without compromising the resource[51]. 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[46] or virtual workspace[47].

Santhanam et al[50] 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. [42], using the original findings by Figueiredo et al[41], 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[42]. To facilitate the creation and deployment of virtual machines the In-Vigo system uses another grid infrastructure tool known as VMPlants[49].

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. [46-48] 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"[47]. 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 [42, 49]. 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[60]. 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[41]. 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 there 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 likely hood 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. [51] 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[57]. 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[51].

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 send 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 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[51].

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[45].

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 AMAZON ELASTIC COMPUTING CLOUD

Recently Amazon released a service for using its computing infrastructure. Amazon Elastic Compute Cloud (Amazon EC2)[62] provides developers with an application programming interface for dynamically creating and managing virtual machines on their large scale web infrastructure. This service can be utilised for grid computing by e-Scientists.

Amazon implements virtualisation within its computing infrastructure however exact details on this are commercially kept in confidence. However, they do provide a standard API for controlling and managing user's virtual machines.

To use this service requires the submission of a virtual machine image. This virtual machine image can be configured with any operating system and software; however there are certain terms and conditions which restrict usage. These are implemented to prevent the misuse of Amazon's infrastructure. Once the virtual machine image is submitted, users can use the API provided or use Amazon's website to interact and control the execution of their machines. Recently, Amazon announced that virtual machine images submitted would now be provided with persistent storage meaning that users can shutdown their machines and restart them at later point.

Usage of Amazon's infrastructure is based on a service costs model. User's pay for what they use, that is, Amazon charges on the amount executions that occur. This cost unit is not directly a single instruction but rather a measure of the computation ability provided for each instance of virtual machine. Users are also charged with any data transfers that occur.

Amazon uses there own virtual machine image format known as Amazon Machine Image (AMI). Amazon provides tools that allow the converting of common virtual machine image formats such as VDMK into their own format. Amazon also provides a library of basic AMI that contain operating system installs.

Primarily designed for web server infrastructure deployment, the Amazon EC2 has also been used in creating distributed applications. A developers' network within Amazon provides knowledge on the type of applications that can be created using the Amazon EC2[63].

4 REFERENCES

1. Jankowski, N.W., *Exploring e-Science: An Introduction.* Computer-Mediated Communication, 2007. **12**(2).
2. Nentwich, M., *Cyberscience - Research in the Age of the Internet*. 2003, Vienna: Austrian Academy of Sciences Press.
3. Wikipedia.org. *e-Science*. [cited; Available from: http://en.wikipedia.org/wiki/E-Science.
4. Kuck, D.J., *High Performance Computing: Challenges for Future Systems*. 1996, New York: Oxford University Press.
5. Wikipedia.org. *Message Passing Interface (MPI)*. 2008 [cited; Available from: http://en.wikipedia.org/wiki/Message_Passing_Interface.
6. Sterling, T.L., *Beowolf Cluster Computing with Linux*. 2002: MIT Press. 496.
7. 500, T., *Top 500 Supercomputer Sites*. 2007.
8. 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.
9. Castagnera, K., et al., *Clustered Workstations and their Potential Role as High Speed Compute Processors*. 1994, NASA Ames Research Center: Moffett Field, CA.
10. Foster, I. and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. 1999, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
11. Foster, I., *The Grid: A new infrastructure for the 21st century science*, in *Physics Today*. 2002, American Institute of Physics.
12. Berstis, V., *Fundamentals of Grid Computing.* IBM Redbooks Paper, 2002.
13. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* Lecture Notes in Computer Science, 2001. **2150**.
14. Anderson, D.P., et al., *SETI@home: An Experiment in Public-Resource Computing.* Communications of the ACM, 2002. **45**(11): p. 56-61.
15. Stanford. *Genome@home*. 2003 [cited; Available from: http://genomeathome.stanford.edu/.
16. Jacob, B., et al., *Introduction to Grid Computing*. 2005, IBM Redbooks.
17. 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.
18. 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.
19. Globus. *Towards Open Grid Services Architecture*. 2008 [cited; Available from: http://www.globus.org/ogsa/.
20. 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.
21. 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.
22. Abramson, D., *Applications Development for the Computational Grid.* Lecture Notes in Computer Science, 2006. **3841**: p. 1 – 12.
23. Sunderam, V.S., *PVM: A Framework for Parallel Distributed Computing.* Concurrency: Practice and Experience, 1990. **2**(4): p. 315-339.
24. Goscinski, W. and D. Abramson, *An Infrastructure for the Deployment of e-Science Applications.*
25. 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.
26. Yu, J. and R. Buyya, *A taxonomy of scientific workflow systems for grid computing.* SIGMOD Rec., 2005. **34**(3): p. 44-49.
27. Altintas, I., et al. *Kepler: An Extensible System for Design and Execution of Scientific Workflows*. in *16th Intl. Conference on Scientific and Statistical Database Management (SSDBM)*. 2004. Santorini Island, Greece.
28. Goldbery, R.P., *Survey of Virtual Machine Research.* IEEE Computer, 1974. **7**(6): p. 34-45.
29. Smith, J.E. and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*. 2005, San Francisco, CA, USA: Elservier Inc.
30. Ferrie, P. *Attacks on Virtual Machine Emulators*. in *Symantec Technology Exchange*. 2007.
31. Bellard, F. *QEMU*. 2008 [cited; Available from: http://bellard.org/qemu/.

32. Bellard, F. *QEMU, a Fast and Portable Dynamic Translator*. in *USENIX 2005 Annual Technical Conference, FREENIX* 2005.
33. VMWare. *VMWare Virtualization*. 2008 [cited 2008 June]; Available from: http://www.vmware.com/virtualization/.
34. Barham, P., et al., *Xen and the art of virtualization.* SIGOPS Oper. Syst. Rev., 2003. **37**(5): p. 164-177.
35. Microsoft. *Virtualization Solutions: Windows Server*. 2008 [cited 2008 June]; Available from: http://www.microsoft.com/virtualization/solution-product-ws.mspx.
36. VMWare. *Virtual Machine Disk Format*. 2008 [cited 2008 June]; Available from: http://www.vmware.com/interfaces/vmdk.html.
37. VMWare, *VMware Virtual Disks - Virtual Disk Format 1.1*. 2007.
38. QEMU. *Disk Images*. 2008 [cited; Available from: http://bellard.org/qemu/qemu-doc.html#SEC15.
39. VMWare. *Virtual Disk Manager*. 2008 [cited 2008 June]; Available from: http://www.vmware.com/support/developer/vddk/VirtualDiskManager.pdf.
40. Tannenbaum, T., et al., *Condor: a distributed job scheduler.* Beowulf cluster computing with Linux, 2002: p. 307-350.
41. 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.
42. 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.
43. Goscinski, W. and D. Abramson, *Motor: A Virtual Machine for High Performance Computing.* 2006: p. 171-182.
44. 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.
45. Joomla! *Grid Appliances*. 2008 [cited; Available from: http://www.grid-appliance.org/.
46. 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.
47. 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.
48. Keahey, K., et al. *Virtual Workspaces in the Grid*. in *Europar*. 2005. Lisbon, Portugal.
49. 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.
50. 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.
51. 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.
52. 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.
53. 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.
54. Ruth, P., et al., *Enabling Autonomic Adaptation of Virtual Computational Environments in a Shared Distributed Infrastructure.*
55. 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.
56. 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.
57. 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.

58. 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.
59. 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.
60. 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.
61. Project, I. *IPOP*. 2008 [cited; Available from: http://www.ipop-project.org/.
62. Amazon. *Amazon Elastic Compute Cloud (Amazon EC2)*. 2008 [cited; Available from: http://www.amazon.com/gp/browse.html?node=201590011.
63. Amazon. *Amazon Web Services Developer Connection*. 2008 [cited; Available from: http://developer.amazonwebservices.com/connect/index.jspa.