



**Universitat Autònoma
de Barcelona**

**Escola d' Enginyeria
Departament d' Arquitectura de Computadors i Sistemes
Operatius
Màster en Computació d'Altes Prestacions**

An Efficient Use of Virtualization in Grid/Cloud Environments.

Memoria del trabajo de “Iniciación a la Investigación.
Trabajo de Fin de Máster” del “Máster en Computació
d'altres Prestacions”, realizada por Arindam Choudhury,
bajo la dirección de Elisa Heymann y Miquel A. Senar.
Presentada en la Escuela de Ingeniería (Departamento
de Arquitectura de Computadores y Sistemas
Operativos).

2011

Iniciación a la investigación. Trabajo de fin de máster del Máster en Computación de Altas Prestaciones.

Título: An Efficient Use of Virtualization in Grid/Cloud Environments.

Realizada por Arindam Choudhury en la Escuela de Ingeniería, en el Departamento de Arquitectura de Computadores y Sistemas Operativos

Dirigida por: Elisa Heymann, Miquel A. Senar

Firmado

Director

Estudiante

Acknowledgement

I would like to express my sincere gratitude to my supervisor Dr. Elisa Heymann and Dr. Miquel A. Senar for their constant encouragement and valuable guidance throughout my research. They has instilled in me a passion towards scientific research that will prove to be vital as I continue to pursue my academic interests. They has prepared me well for this thesis work and has worked hard to promote my work. I am thankful for their continual support, encouragement and invaluable suggestions throughout this thesis work. They not only provided me help whenever needed, but also provided me the resources required to complete this thesis report on time.

I am also grateful to Dr. Alain Roy, Department of Computer Sciences, University of Wisconsin–Madison, for his valuable support and advice in installation, working and understanding the Condor system.

I would also like to thanks users of condor-users mailing list specially Dr. Mark Calleja, Computing Science group, University of Cambridge and Dr. Matthew Farrellee, Condor Team for their help to understand Condor System and Condor VM Universe.

My heartfelt thanks go to my parents, to my brothers and to my friends, especially Cesar, Claudio and Guifre. This work could not have been accomplished without their support.

Abstract

Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational resources. Grid enables access to the resources but it does not guarantee any quality of service. Moreover, Grid does not provide performance isolation; job of one user can influence the performance of other user's job. The other problem with Grid is that the users of Grid belong to scientific community and the jobs require specific and customized software environment. Providing the perfect environment to the user is very difficult in Grid for its dispersed and heterogeneous nature. Though, Cloud computing provide full customization and control, but there is no simple procedure available to submit user jobs as in Grid.

The Grid computing can provide customized resources and performance to the user using virtualization. A virtual machine can join the Grid as an execution node. The virtual machine can also be submitted as a job with user jobs inside. Where the first method gives quality of service and performance isolation, the second method also provides customization and administration in addition.

In this thesis, a solution is proposed to enable virtual machine reuse which will provide performance isolation with customization and administration. The same virtual machine can be used for several jobs. In the proposed solution customized virtual machines join the Grid pool on user request. Proposed solution describes two scenarios to achieve this goal. In first scenario, user submits their customized virtual machine as a job. The virtual machine joins the Grid pool when it is powered on. In the second scenario, user customized virtual machines are preconfigured in the execution system. These virtual machines join the Grid pool on user request.

Condor and VMware server is used to deploy and test the scenarios. Condor supports virtual machine jobs. The scenario 1 is deployed using Condor VM universe. The second scenario uses VMware-VIX API for scripting powering on and powering off of the remote virtual machines. The experimental results shows that as scenario 2 does not need to transfer the virtual machine image, the virtual machine image becomes live on pool more faster. In scenario 1, the virtual machine runs as a condor job, so it easy to administrate the virtual machine. The only pitfall in scenario 1 is the network traffic.

Keywords: Grid computing, Cloud computing, Virtualization

Resumen

Grid es una infraestructura de hardware y software que proporciona fiabilidad, acceso constante, generalizado y barato a recursos computacionales de alto nivel. Grid permite el acceso a los recursos pero no garantiza la calidad del servicio. Además, Grid no proporciona un aislamiento de rendimiento; el job de un usuario puede influir en el rendimiento del job de otro usuario. Otro problema de la tecnología Grid es que los usuarios del Grid pertenecen a una comunidad científica y sus jobs requieren un entorno específico y personalizado. Proporcionar el entorno ideal es muy difícil por la naturaleza dispersa y heterogénea del sistema Grid. Sin embargo, los sistemas Cloud computing sí proporciona una personalización y control completo, pero no existen procesos simples para la entrega de jobs como ocurre en los entornos Grid.

La computación en Grid proporciona recursos y rendimiento personalizados al usuario mediante la virtualización. Una máquina virtual puede unirse al Grid como nodo de ejecución. Además, una máquina virtual puede ser entregada como un job, con jobs de usuario en su interior. Mientras que el primer método proporciona calidad de servicio y aislamiento de rendimiento, el segundo método además proporciona personalización y administración.

En esta tesis, se propone una solución para habilitar la reutilización de máquinas virtuales que han de proporcionar aislamiento de rendimiento y capacidades de administración. La misma máquina virtual puede ser utilizada para múltiples jobs. En la solución propuesta, las máquinas virtuales se una al pool del Grid a petición del usuario. Se describen dos escenarios para conseguir esta meta. En el primer escenario, el usuario entrega su máquina virtual personalizada como un job. Al iniciarse la máquina virtual, ésta se une al pool del Grid. En el segundo escenario, máquinas virtuales personalizadas para el usuario son preconfiguradas en el entorno de ejecución. Estas máquinas virtuales se unen al pool del Grid bajo demanda del usuario.

Se ha utilizado Condor y VMware server para desplegar y evaluar los escenarios. Condor soporta jobs de tipo máquina virtual. El escenario 1 se despliega utilizando Condor VM universe. El segundo escenario utiliza la API VMware-VIX para activar la máquina virtual remota mediante scripts. Los resultados experimentales muestran que el escenario 2 no necesita transferir la imagen de la máquina virtual. La imagen de la máquina virtual se

activa en el pool de forma más rápida. En el escenario 1, la máquina virtual se ejecuta como un job de Condor, y resulta fácil administrar la máquina virtual. El único problema encontrado en el escenario 1 es el tráfico de red.

Palabras Clave: Grid computing, Cloud computing, Virtualización

Resum

Grid és una infraestructura hardware i software que proporciona fiabilitat, accés constant, generalitzat i econòmic als recursos computacionals d'alt nivell. Grid permet l'accés als recursos però no garanteix la qualitat de servei. A més, Grid no proporciona un aïllament del rendiment; el job d'un usuari pot influir en el rendiment del job d'un altre usuari. Un altre problema de la tecnologia Grid és que els usuaris del Grid pertanyen a una comunitat científica i els jobs requereixen un entorn específic i personalitzat. Proporcionar l'entorn perfecte per l'usuari és molt difícil en els entorns Grid degut a la seva naturalesa dispersa i heterogènea. A diferència, Cloud computing sí que proporciona una personalització i control complet, no obstant, no existeixen processos simples per l'entrega de jobs com passa als entorns Grid.

La computació en Grid proporciona recursos i rendiment personalitzats per l'usuari mitjançant la virtualització. Una màquina virtual pot unir-se al Grid com un node d'execució. També, una màquina virtual pot ser entregada com un job amb jobs d'usuari al seu interior. Mentre que el primer mètode proporciona qualitat de servei i aïllament del rendiment, el segon mètode, a més, proporciona personalització i administració.

En aquesta tesi, es proposa una solució per habilitar la reutilització de màquines virtuals que han de proveir aïllament del rendiment i capacitat d'administració. La mateixa màquina virtual pot ser emprada per múltiples jobs. En la solució proposada, les màquines virtuals s'uneixen al pool del Grid sota demanda d'usuari. Al primer escenari, l'usuari entrega la seva màquina virtual personalitzada com a job. Al iniciar la màquina virtual, aquesta s'uneix al pool del Grid. En el segon escenari, màquines virtuals personalitzades per a l'usuari són preconfigurades en l'entorn d'execució. Aquestes màquines virtuals s'uneixen al pool del Grid sota demanda de l'usuari.

S'han emprat Condor i VMware server per desplegar i avaluar els escenaris. Condor suporta jobs del tipus màquina virtual. L'escenari 1 es desplega mitjançant Condor VM universe. El segon escenari utilitza la API VMware-VIX per a activar la màquina virtual remota mitjançant scripts. Els resultats experimentals mostren que l'escenari 2 no necessita transferir la imatge de la màquina virtual. La imatge de la màquina virtual s'activa en el pool de forma més ràpida. En l'escenari 1, la màquina virtual s'executa com a job de

Condor, i resulta fàcil l'administració d'aquesta màquina virtual. L'únic problema trobat a l'escenari 1 és el tràfic de xarxa.

Paraules Clau: Grid computing, Cloud computing, Virtualització

Index.

Acknowledgement.....	i
Abstract	iii
Resumen.....	v
Resum.....	vii
Index.....	ix
List of Figures	xiii
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Objective.....	2
1.3 Organization of Thesis.....	2
Chapter 2 State of the Art	3
2.1 Grid Computing.....	3
2.1.1 History of Grid	5
2.1.2 Grid Types	7
2.2 Grid Architecture.....	8
2.2.1 Grid Protocol Architecture	8
2.2.2 OGSA: Open Grid Services Architecture.....	10
2.3 Cloud Computing	12
2.3.1 Cloud Computing Stack	14
2.3.2 Market Oriented Cloud Architecture.....	16
2.4 Virtualization	18
2.4.1 Benefits of Virtualization	18
2.4.2 Types of Virtualization.....	19
2.5 Benefits of Virtualization in Grid Computing:.....	21
2.6 Grid Virtualization Requirements	22
2.6.1 High Performance Computing.....	22
2.6.2 Grid middleware for virtual machines.....	22
2.6.3 Virtual Machine Image Building.....	23

2.7	Virtualization in Grid Computing.....	23
2.7.1	Globus Virtualization Workspace	23
2.7.2	CernVMs	24
2.7.3	Condor VM Universe	25
2.8	Virtual Machine Management	27
2.8.1	Xen Hypervisor	27
2.8.2	VirtualBox	27
2.8.3	VMware	28
Chapter 3 Proposed Solution.....		29
3.1	Proposed Solution.....	29
3.1.1	Scenario 1	30
3.1.2	Scenario 2	31
3.2	Application Used	32
3.3	Deployment Details:	34
3.3.1	Scenario 1:	34
3.3.2	Scenario 2	35
Chapter 4 Experimentations and Results		37
4.1	Common Configuration	37
4.1.1	Condor Installation	37
4.1.2	Virtual Machine Image Creation:	39
4.2	Scenario 1	40
4.2.1	Condor VM Universe Configuration.....	40
4.2.2	Virtual Machine Image Creation and Submitting VM Universe Jobs	41
4.2.3	Making the Virtual Machine Private	42
4.2.4	Submitting jobs the Scenario 1	42
4.3	Scenario 2	43
4.3.1	Configuring Execution Node:.....	43
4.3.2	Making the Virtual Machine node Private:	44
4.3.3	Powering on the virtual machine:.....	45
4.3.4	Submitting job	45
4.3.5	Powering off the Virtual machine:	46
4.4	Evaluation of results	46

Chapter 5 Conclusions and Future Work	49
5.1 Conclusions	49
5.2 Future Work.....	50
Bibliography.....	53

List of Figures

Figure 2-1 Generic View of World-Wide Grid (WWG) Computing Environment [10].	4
Figure 2-2 The Layered Grid Architecture and its Relationship with TCP/IP Protocol Architecture [11].	8
Figure 2-3 The OGSA Architecture [18].	11
Figure 2-4 Benefits of Cloud Computing [20].	12
Figure 2-5 Cloud Computing Stack [21].	14
Figure 2-6 High-level Market-oriented Cloud Architecture [22].	16
Figure 2-7 Type 1 Hypervisor.	19
Figure 2-8 Type 2 Hypervisor.	20
Figure 2-9 Globus Virtual Workspace [28].	23
Figure 2-10 Building Blocks of CernVM [31].	24
Figure 2-11 Condor VM Universe [32].	26
Figure 3-1 Scenario 1.	30
Figure 3-2 Flowchart of Scenario 1.	30
Figure 3-3 Scenario 2.	31
Figure 3-4 Flowchart of Scenario 2.	32
Figure 3-5 Condor Pool and Daemons [32].	32
Figure 4-1 Condor Pool.	37
Figure 4-2 Condor Pool with Virtual Machine Node.	40
Figure 4-3 Virtual Machine Starting Up.	41
Figure 4-4 Job Submission in Scenario 1.	43
Figure 4-5 Powering On the Virtual Machine.	44
Figure 4-6 Job Submission Errors.	45
Figure 4-7 Network Bandwidth Usage.	47

Chapter 1 Introduction

Grid [1] [2] offers an optimal solution to problems requiring large storage and/or processing power. Grid provides direct access to computers, data, software and many other resources. Sharing between these resources is highly controlled and done with consensus of both resource providers and consumers.

Cloud computing [3] [4] offers services to the users. Cloud uses virtualization [5] to provide resources to the users. Virtualization enables efficient access and management of resources to reduce operations and systems management costs while maintaining needed capacity. Virtualization optimizes workload and on one host system multiple virtual machines can be run.

1.1 Motivation

Though grid computing provides access to high end computing resources, it did very little to ensure the quality of service [6]. Most of the Grid middleware does not support performance isolation also. Moreover Grid is mainly used by scientific community and their job execution needs fine-tuned execution environment. But providing perfect execution environment on remote execution system is a trivial task.

Grid computing can use virtualization to provide quality of service and performance isolation. Users can submit their jobs on own custom virtual machines which will provide perfect execution environment. Currently, virtual machines are deployed in Grid in two ways. In the first the virtual machine joins the grid pool as execution node. This can provide quality of service and performance isolation but this cannot guarantee custom execution environment. In the second way, the user submits their custom virtual machine as a job. The user puts his jobs inside virtual machine and configures it to run the jobs on system boot up. Grid middleware takes care of virtual machine image transfer, configuring the virtual machine for the remote execution system and powering it on. The virtual machine needs to be configured so it automatically shutdown on job completion. This signals the Grid middleware that the job completes. Then Grid middleware transfers the modified virtual machine image to the user. This method gives complete flexibility to the user to administrate and create own custom job environment in addition of quality of service and performance isolation.

1.2 Objective

Grid computing allows users to submit their custom virtual machine as a job. The user has to submit virtual machine each time he wants to run his jobs. The virtual machine images are huge so transferring these images can create bottleneck in network traffic. So, the objective is to propose a solution where user custom virtual machines can be reused by the user. This can be done by allowing user created virtual machines to join the Grid pool. The solution can be achieved in two ways; in first the user will submit the virtual machine node as a job. When the virtual machine will be powered on, it will join the available Grid pool. The other way takes advantage of the fact that group of user uses same execution environment, so the requisite virtual machine can be obtained and configured in the execution node. These virtual machines can join the condor pool on user demand.

The proposed solution is deployed using Condor. Condor pool is created using three systems. VMware Server 1 is used as the virtualization hypervisor.

1.3 Organization of Thesis

Chapter Two gives an overview of Grid computing, Cloud computing and virtualization. It discusses various aspects of Grid computing and gives historical pathway through which Grid computing has evolved. It also discusses about the architecture of Grid computing: the protocol architecture and the OGSA architecture. It also describes Cloud computing, benefits of using Cloud computing, the Cloud computing stack, the Cloud architecture and how Cloud computing is greatly benefitted by use of virtualization. Then virtualization is discussed. The different types of virtualization techniques are discussed and also the benefit we can get from virtualization.

Chapter Three discusses the proposed solution to enable reuse of virtual machines in Grid computing. It describes the two different kind of scenario can be used to deploy the solution. It details the applications used for deployment. It also describes all the deployment issues.

Chapter Four details the experimentation and results. It details the configuration required to deploy both scenarios. It also discusses the results obtained from both scenarios.

Chapter Five concludes the thesis and discusses about the future work.

Chapter 2 State of the Art

Today's scientific community needs enormous processing power and storage capacity. Grid computing [1] [7] emerges in this context and enables access to powerful and heterogeneous resources which are physically dispersed in various geographical locations. Cloud computing [8] emerges in an e-business context and provides an illusion of infinite computing resources available on demand. Cloud computing uses virtualization to allocate, de-allocate, and to migrate computing resources. Both Grid and Cloud computing consists of customized services which delivers computing power as a utility.

The following sections provide detailed discussion about Grid computing, Cloud computing and virtualization.

2.1 Grid Computing

Performance of computing can be improved by using a better algorithm, using a faster computer or dividing the calculation among multiple computers. Algorithms can be fine-tuned to a limit. Faster computers are costly and there is also a limitation in speed of computer. The best way to improve the performance of computing is dividing the calculation among multiple computers. Desktop computers (like most others) are only utilized about 5% - the CPU stays idle most of the time [9]. Grid computing uses this idle time and links many machines together to perform massive tasks that previously only super-computers could do. Traditional distributed computing can be characterized as a subset of Grid computing. Distributed Computing normally manages or pools computer systems which are limited in their memory and processing power. On the other hand, Grid computing is concerned with efficient utilization of a pool of heterogeneous systems with optimal workload management utilizing computational resources (servers, networks, storage, and information) acting together to create large pool of computing resources.

The computational Grid can be compared to electric power Grid. As power Grid distributes electricity to dispersed remote places, Grid computing enables distribution of computing resources. Grid computing has been researched in universities and institutions for long time. With creation of sophisticated policies and standards, Grid computing is spreading and used in many scientific and engineering organizations. Engineers, biologist, earth

scientists, high energy physicists, animation artist are using Grid computing for solving compute or data intensive problems. Figure 2-1 shows generic view of World-Wide Grid.

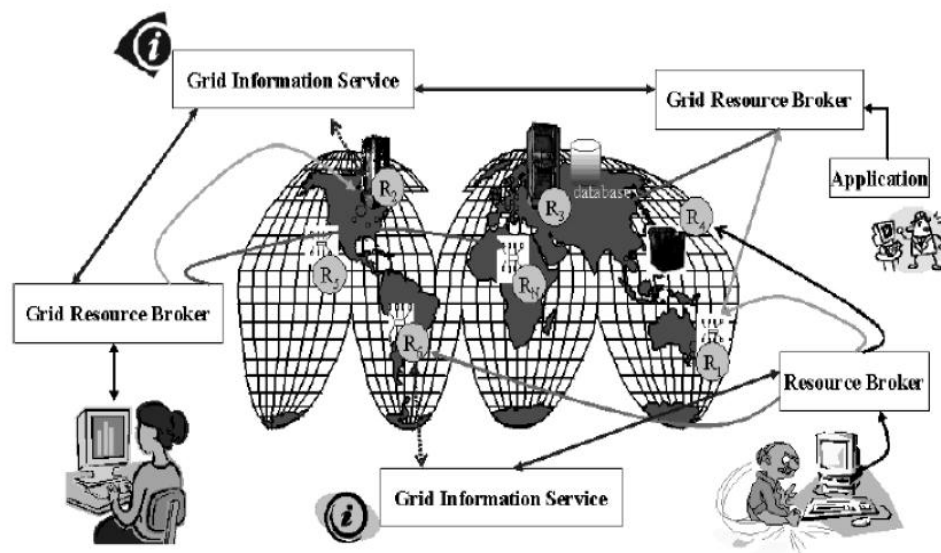


Figure 2-1 Generic View of World-Wide Grid (WWG) Computing Environment [10].

Grid Computing provides direct access to computers, data, software and many other resources, which are heterogeneous in nature and lie in different administrative domains [11]. The sharing policies of these resources are different. To accommodate these resources, the Grid has to support a common mechanism for communication between these computers. The sharing must be highly controlled and done with the consensus of both resource consumers and resource providers. Virtual Organizations are formed using these sharing rules. An organization can participate in two or more VOs.

Carl Kesselman and Ian Foster have given a definition for Grid computing in the book “The Grid: Blueprint for a New Computing Infrastructure.” They define Grid as: “A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.”

Ian Foster also defined three point checklists for Grid computing [12], which are:

- Coordinates resources that are not subject to centralized control.
- Uses standard, open, general-purpose protocols and interfaces.
- Delivers nontrivial qualities of service.

Security is an important aspect of Grid computing. Authentication, authorization, encryption and non-repudiation also used in Grid computing. Besides these Grid uses following security mechanisms [13]:

- Single sign-on: support authentication to a large number of Grid resources on behalf of the user or resources. It creates temporary credentials (secure proxy) that provide authentication.
- Delegation: one Grid entity act on behalf of another Grid entity. Delegation must be handled carefully to resist the misuse.
- Community authorization: using this virtual organizations define policies to access resources.
- Secure execution: provide a service using which untrusted applications can run on trusted environment.

2.1.1 History of Grid

A decade ago, computers were expected to use local resources and data. But the scenario changed when system got interconnected through network. An expectation to use the ideal computing resources and data storage to fulfill a range of application needs had been generated. The designing, building and deploying of distributed computing has been researched over many years. These researchers implemented middleware, libraries and tools that allow cooperative use of distributed resources. This approach of computing has been given different names, such as metacomputing, scalable computing, global computing and lately as Grid computing [14] [15].

- The First Generation of Grid Computing: In early to mid-1990s Grid efforts started to link the supercomputing sites. The FAFNER and I-WAY are two well-known projects that show the pathway to build Grid. FAFNER was built to facilitate factoring via web of RSA factoring parallel algorithm. I-WAY helped to unify the resources of large US supercomputing centers. I-WAY used I-POP as gateway servers. I-POP software environment is used in the servers to overcome issues concerning heterogeneity, scalability, performance and security. I-POP servers are accessible via the Internet. The well-known Grid middleware Globus Toolkit is derived from the I-WAY project.
- The Second Generation of Grid Computing: The three main issues that must be addressed in Grid are scalability, heterogeneity and adaptability. Grid middleware

provide abstraction over resource heterogeneity and makes the Grid scalable and adaptable. Usage of common standards also provides a solution to heterogeneity. Globus [16], a widely used Grid middleware, enables the construction of computational Grid. It supports wide range of application and programming paradigms. Globus is modular and constructed as a layered architecture. Globus provides GRAM to allocate, monitor and control computational resources. Globus uses GridFTP for data access. Globus further supports authentication and related security support, Lightweight Directory Access Protocol (LDAP), Global Access to Secondary Storage (GASS), Globus Advanced Reservation and Allocation (GARA).

Legion another popular Grid middleware, provide infrastructure to Grid computing. But it differs from Globus as it encapsulates all components as objects. The OMG (Object Management Group) in 2001 introduced CORBA in Grid through “Software Service Grid Workshop”. Jini & RMI is a CORBA based distributed computing environment.

In Grid, resources need to be discovered, monitored, and scheduled. The jobs submitted needed to be queued and batched. Condor, Portable Batch System (PBS), Sun Grid Engine (SGE), Load Sharing Facility (LSF) are some of the batch and scheduling system created in this generation Grid computing. Nimrod/G is a Grid enabled resource management and scheduling system. Nimrod/G supports user defined budget constraints and deadlines.

In the second generation of Grid computing, the core technology for building a Grid evolved. Besides that additional tools were also developed to support resource and job management.

- The Third Generation of Grid Computing: Third generation of Grid computing aims to the reuse of existing resources and information systems, and to assemble these resources. The use of service-oriented model and metadata are two main characteristics of third generation Grid computing. The concept of “Virtual Organization” and “Distributed Collaboration” were adopted. This generation of Grid implements an autonomic system which can configure and reconfigure itself dynamically, implement open standards, recover from malfunction and optimize use of resources. It used service-oriented architecture like web-services, Open Grid Services Architecture (OGSA) , and agents. Web services support third generation

Grid requirements as it supports service-oriented approach and use standards to facilitate the information aspects. OGSA supports creation and maintenance of Virtual Organizations. Agent based computing provide autonomy to the dynamically changing Grid environment.

The evolution of Grid is still in progress. Next sub section discusses about different kind of Grids.

2.1.2 Grid Types

Grid provides different type of services and resources [17] Depending upon the design objectives and target applications Grid computing can be broadly categorized in different types. The different types of Grids are discussed below:

Grids can be categorized by the type of resources they share. The Grids who shares different type of resources, they fall into more than one categories of Grid given below:

- **Computational Grids:** In this type of Grid primarily CPU resources are shared. Examples of computational Grids are TeraGrid, SETI@home, and Sun Grid Compute Utility etc.
- **Data Grids:** Data resources such as result of experiments are shared between users. Data Grids handle large amount of distributed data. Examples of data Grids are QCDGrid and LHC computing Grid.
- **Storage Grids:** Access to enormous amount of storage space is provided by storage Grid. One of the biggest examples of storage Grid is Amazon S3.
- **Equipment Grids:** Equipment Grids share accesses to physical resources such as astronomical telescope are shared in eSTAR project.

Grid can be classified by the geographically distribution of their resources.

- **Internet-scale Grid:** In this type of Grid anyone with access to internet can participate. SETI@home and World Community Grid are examples of Internet-scale Grids.
- **Virtual Organization-scale Grids:** A VO-scale Grid contains several academic or corporate entities. Most Grids fall into this category e.g. TeraGrid, QCDGrid.
- **Local Grids:** Local Grids are deployed within one organization. No other organization has access to this Grid. For example, the render farms used to produce animated films such as Toy Story are a form of local Grid.

Services provided by a Grid are the most important thing. Data Grid provided access to result of a large physics experiment. Some Grid may not be able run some applications due to platform or licensing restrictions. Therefore, as Grid becomes more prevalent, more initiatives such as Network.com which provides a catalogue of applications that can be run on the Sun Grid computing facility can be seen.

At the moment, the most prevalent types of service offered by Grids are forms of graphical rendering, scientific simulations and web applications.

2.2 Grid Architecture

Grid needs special architecture to cope up with heterogeneous resources and technologies. The architectures are discussed below:

2.2.1 Grid Protocol Architecture

The layered architecture of Grid computing [11] is built on the TCP/IP protocol and services. The layers here are conceptual and do not imply constraints. Common mechanisms, interfaces, schemas, and protocols are defined in each layer, using which sharing relationship can be established, and computational power and data storage can be shared over network. Figure 2-2 below shows the different layers of Grid architecture and shows its relationship with TCP/IP protocol architecture:

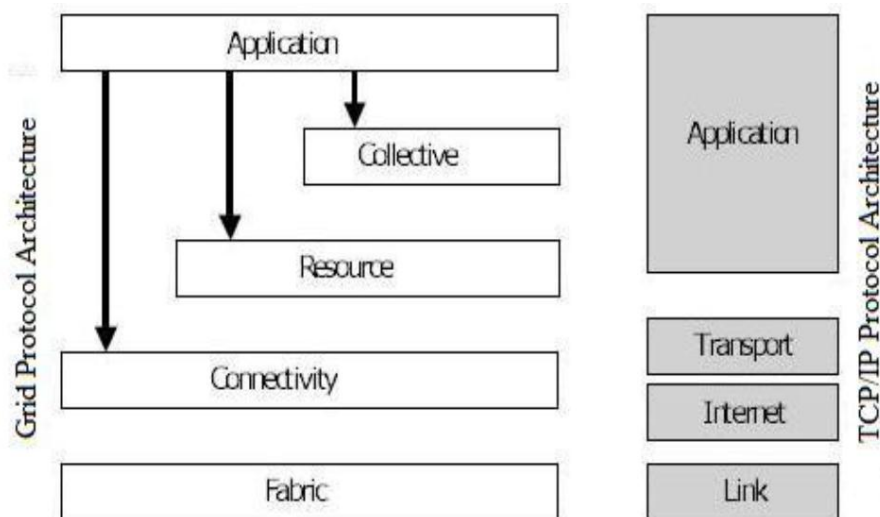


Figure 2-2 The Layered Grid Architecture and its Relationship with TCP/IP Protocol Architecture [11].

The Grid architecture consists of five layers. These layers are Fabric, Connectivity, Resource, Collective and Application. The brief discussions of these layers are given below:

- **Fabric Layer:** Fabric layer [11] defines set of operations to be performed on local resources to initiate sharing relationship at higher level. These operations are local and resource specific. The more sophisticated operations implemented in this layer, the more flexible relationship can be obtained in the higher level. Fabric layer should implement function to report structure, state and capabilities of the local resources, and resource management mechanisms.
- **Connectivity Layer:** Connectivity layer [11] defines the core communication and authentication protocols. Communication protocols are drawn from the Internet, transport and application layers of TCP/IP protocol stack. Authentication protocols provide security aspects of Grid resource sharing. Authentication protocols also supports for communication protection.
- **Resource Layer:** The resource layer [11] defines protocol, which are built on connectivity layer protocol, for secure negotiation, control and monitoring of individual resources. Protocol of this layer calls functions of fabric layer to access and control local resources. These protocols can be divided into information protocol, used to obtain information about state and structure, and management protocol which used to negotiate resource sharing specifying resource requirements and operation to be performed.
- **Collective Layer:** The collective layer [11] captures interaction between collections of resources where resource layer focused on single resource. This layer provide persistent services with associated protocols to support directory services to discover existence and properties of virtual organization and resources, co-allocation, scheduling, and brokering services, monitoring and diagnostics services, and many other services to facilitate data replication, community authorization, workload management etc.
- **Application Layer:** Application layer [11] is the final layer of Grid architecture and it comprises of the user applications that operate within a virtual organization.

Grid protocol architecture defines the protocol should be used in the various layers. But an architecture is needed which can deal with the various services provided by the Grid.

OGSA is such an architecture which defines the creation of services and how the services can be manipulated.

2.2.2 OGSA: Open Grid Services Architecture

Open Grid Services Architecture (OGSA) [18] is a proposed set of standards for ensuring quality of service across a Grid computing network. It designed for service oriented Grid computing by Global Grid Forum (GGF). The objectives of OGSA are to manage resources across distributed heterogeneous platforms, delivery of seamless Quality of Service, and providing a common base for autonomic management solutions. The OGSA defines open and published interfaces. To facilitate interoperability Grids must use standard interfaces and protocols. OGSA extends web services for Grid. It exploits industry standard integration technologies.

OGSA extends web services for Grid computing. OGSA manages resources in Grid [18] by providing:

- Scalability: OGSA provides scalability through hierarchical management of Grid resources.
- Interoperability: OGSA use standard interfaces between heterogeneous resources. General IT management standards are also standardized.
- Security: security is deployed by using authentication, authorization, access control, access policy. Management also ensures its own integrity by using security mechanism on management tasks.
- Reliability: reliability is achieved by not forcing a single point of failure. Managers are allowed to manage multiple resources and manageable resources are allowed to manage by multiple manager.

The OGSA architecture is shown below in the Figure 2-3. The OGSA architecture comprises of four main layers. Short descriptions of these layers are given below:

- Resources- Physical resources and logical resources: In Grid, resources are not limited to just processor. Resources like processor, servers, storage and networks are physical resources. Logical resources are above the physical layer and provide additional function by virtualizing and aggregating the physical resources. General purpose middleware provide these abstract services on top of physical Grid.
- Web Services layer: In the second layer of OGSA both physical and logical resources are modeled as services using Open Grid Services Infrastructure (OGSI).

OGSI built on standard Web services technology and exploits mechanisms of Web services like WSDL and XML to specify standard interfaces, behaviors and interactions for Grid resources.

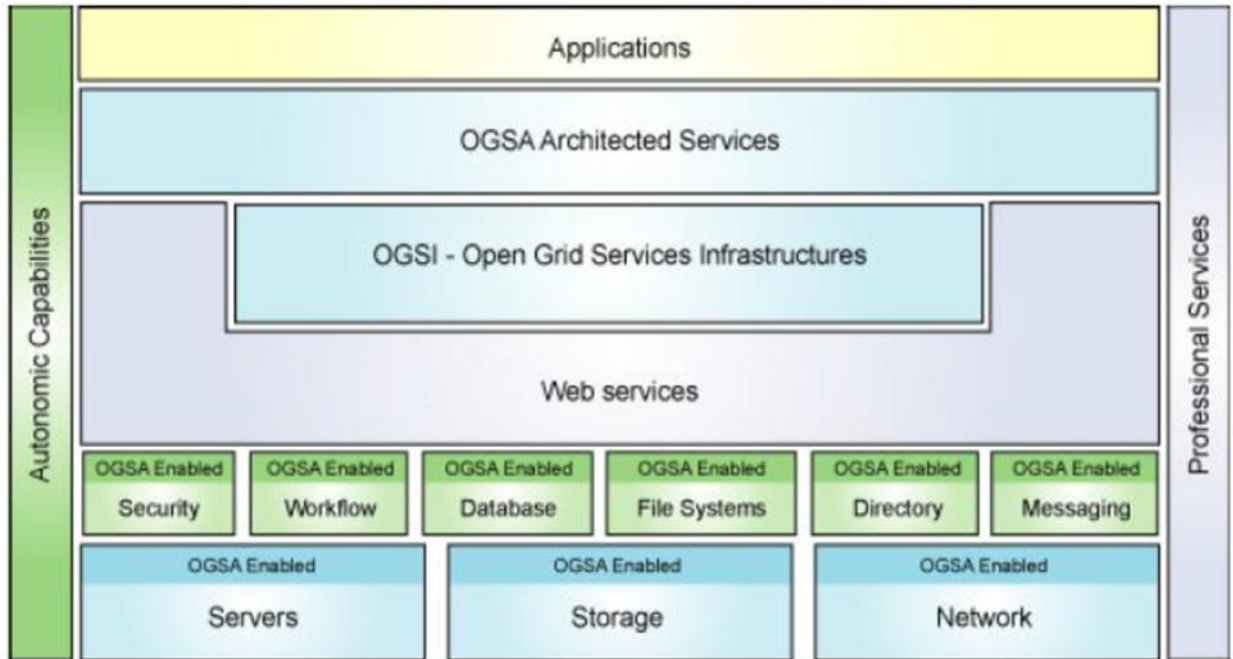


Figure 2-3 The OGSA Architecture [18].

- OGSA architected Grid services layer: In this layer many Grid services, architected with OGSI extensions, are defined in the area of program execution, data services, and core services.
- Grid Application Layer: In this last layer of OGSA, the applications that use OGSA architected Grid services of level three are defined.

Grid shares different kind of resources and exploits them. Grid is not static, any Grid node or resource can join or leave at any time. So, a mechanism is needed to manage these nodes or resources to enable proper sharing. Grid Resource Management is designed to fulfill this goal.

Though Grid has resource management deployed to manage resources, sharing and interaction between heterogeneous resources may lead to errors and failures in Grid. Thus a monitoring system is essential. To achieve high-performance Grid, it is critical to monitor and manage the system. Monitoring data determine the source of performance problems and help to tune the system for better performance. This data is also used by fault detection and recovery system to determine if a server is down, these data also help to make decision like restarting the server or redirecting service requests elsewhere.

Prediction model use these data to predict performance, schedulers use these predictions to determine which resource to use.

Next Section introduces Cloud computing. Cloud computing also offers access to remote high end resources but the policies and strategies used are different.

2.3 Cloud Computing

Though there is no proper definition of the Cloud computing, in [19] Cloud computing defined as: Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized Service Level Agreements.

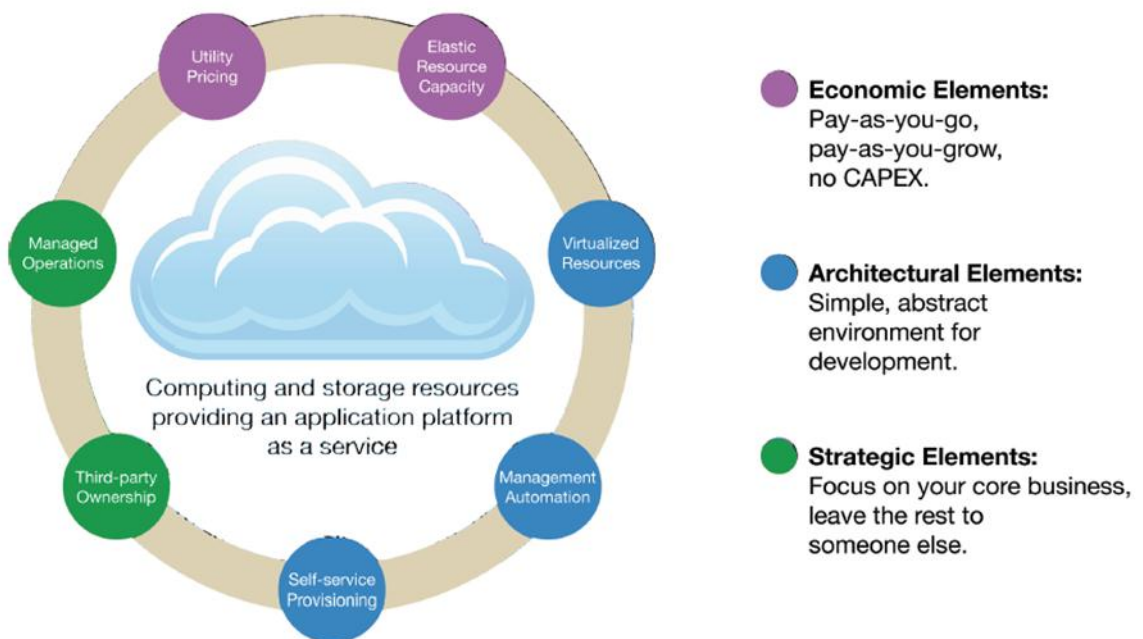


Figure 2-4 Benefits of Cloud Computing [20].

Figure 2-4 depicts the benefits of Cloud computing [20] which are discussed in detail below:

- **Utility Pricing:** users of Cloud computing consume computing and storage services on demand. They have to pay only for the computing power, storage or network

bandwidth they have consumed. They only pay Operating Expenses (OPEX) rather than Capital Expenditures (CAPEX).

- **Elastic Resource Capacity:** in cloud users are not tapped with a fixed set of resources. User can add or remove capacity at will. This gives users the ability to scale up or down when needed and an illusion of infinite resources. This also eliminates any up-front commitment as the resources are provided on demand.
- **Virtualized Resources:** Cloud computing would not be possible without virtualization. In Cloud, resources are shared from a common infrastructure by multiple groups of users. This only can be achieved through virtualization. Cloud computing creates virtual slices of resources and storage devices to fulfill perfectly the specific need of multiple users.
- **Management Automation:** Cloud computing is highly standardized. Most Cloud computing usually standardized a single kind of CPU, a single hypervisor or virtual machine monitor, a single operating system and a single database. The standardization helps to reduce operating cost dramatically through aggressive management automation.
- **Self-service Provisioning:** self-service provisioning is the fundamental difference between Application Service Provider and Cloud computing. Because of self-service provisioning in Cloud, users can provision resources with just some mouse clicks. Dedicated servers did not have to be allocated to every user.
- **Third-party Ownership:** Cloud computing is also a new form of outsourcing. The user does not have to take care about any deployment details of the datacenter, the cloud is hosted on. All the technical difficulties are burden of the cloud owner. Cloud computing is all about the transfer of ownership for such resources to a third-party that specializes in their deployment.
- **Managed Operations:** Cloud computing just does not provide IT infrastructure managed by a third-party, but it also manages software upgrades, data backups and countless other tasks required to manage mission-critical business applications. The service is provided to the user according to a well-defined Service Level Agreements.

2.3.1 Cloud Computing Stack

Cloud computing can be categorized according to the abstraction level of capability provided and service model of the providers. Figure 2-5 depicts the Cloud computing stack [21]:

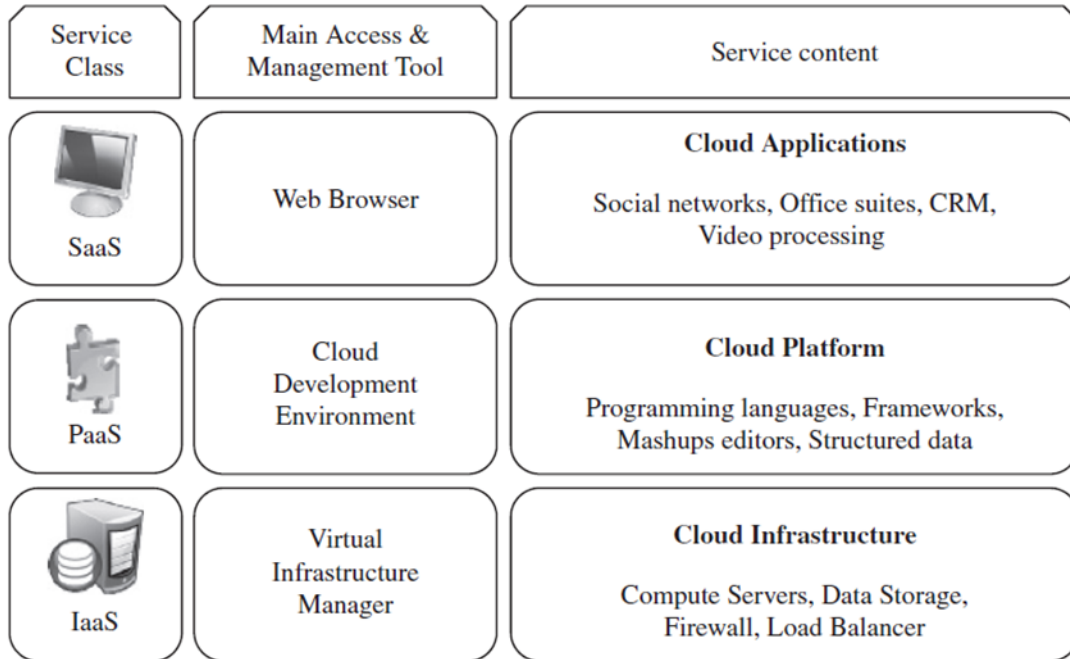


Figure 2-5 Cloud Computing Stack [21].

- Infrastructure as a Service (IaaS): Infrastructure as a Service abstracts underlying hardware and outsources the equipment used to support operations including storage, hardware, servers and networking components.

Characteristics and components of IaaS include:

- Utility computing service and billing model.
- Automation of administrative tasks.
- Dynamic scaling.
- Desktop virtualization.
- Policy-based services.
- Internet connectivity.

Infrastructure as a Service is sometimes referred to as Hardware as a Service (HaaS).

- **Software as a Service (SaaS):** Software as a Service (SaaS) is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the Internet. SaaS is closely related to the Application Service Provider and on demand computing software delivery models. There are two different delivery models for SaaS. In the hosted application management model a provider hosts commercially available software for customers and delivers it over the Web. In the software on demand model, the provider gives customers network-based access to a single copy of an application created specifically for SaaS distribution.

Benefits of the SaaS model include:

- Easier administration
 - Automatic updates and patch management
 - Compatibility: All users will have the same version of software.
 - Easier collaboration, for the same reason
 - Global accessibility.
- **Platform as a Service (PaaS):** PaaS allows the customer to rent virtualized servers and associated services for running existing applications or developing and testing new ones. Users do not have to deal with the cost and complexity of buying or managing the underlying hardware or software. PaaS provides all the support for building web applications or services entirely available from the Internet. PaaS offerings may include facilities for application design, application development, testing, deployment and hosting as well as application services such as team collaboration, web service integration and marshaling, database integration, security, scalability, storage, persistence, state management, application versioning, application instrumentation and developer community facilitation. These services may be provisioned as an integrated solution over the web.

Benefits of the SaaS model include:

- Services to develop, test, deploy, host and maintain applications in the same integrated development environment.
- Web based user interface creation tools
- Multi-tenant architecture
- Integration with web services and databases
- Support for development team collaboration

- Utility-grade instrumentation

Cloud providers need to meet specific quality of service to meet their objective and sustain their operation. Cloud does not use traditional system-centric resource management system. Cloud uses a market oriented architecture which is described in the next section.

2.3.2 Market Oriented Cloud Architecture

Market oriented architecture is used in Cloud computing to supply resources to the users on demand. This architecture provides feedback to users and the Cloud provider in terms of economic incentives. It also promotes Quality of Service-based resource allocation mechanisms.

Figure 2-6 shows the High-level market-oriented Cloud architecture [22]:

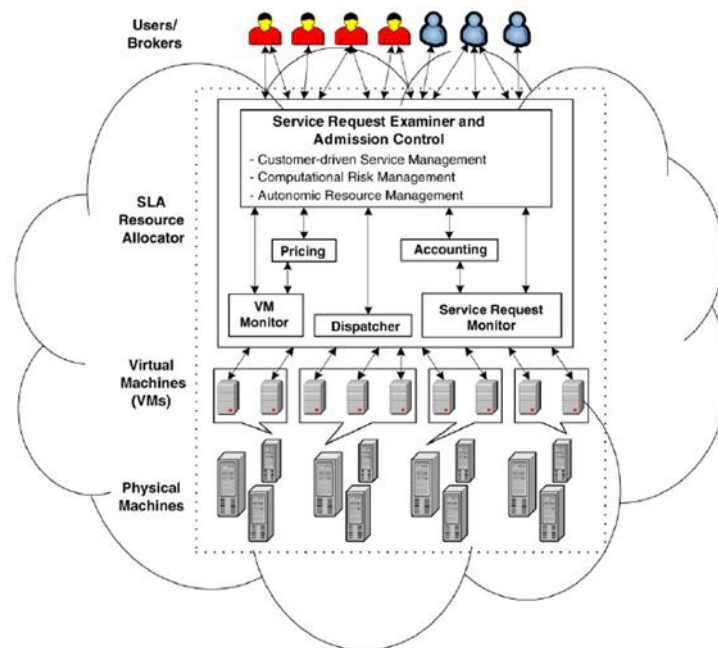


Figure 2-6 High-level Market-oriented Cloud Architecture [22].

The architecture made of four main entities. Entities are described below:

- Users/Brokers: users or brokers can submit their service request to the cloud from anywhere in the world.
- SLA Resource Allocator: Service Level Agreements Resource Allocator acts as the interface between the Cloud provider and external users or brokers. The interaction mechanism of SLA Resource Allocator is discussed below:

- Service Request Examiner and Admission Control: checks submitted request for quality of service requirements to determine if the request is acceptable. It ensures that the request submitted should not overload the available resources. It uses resource availability and workload processing matrices to make resource allocation decisions effectively. Resource availability is supplied from Virtual Machine Monitor mechanism and workload processing is supplied from Service Request Monitor mechanism. It allocates the request to virtual machines and then assigns resources to the allocated virtual machines.
- Pricing: calculates how the submitted request will be charged. The request can be charged based on submission time (peak/off-peak), pricing rates (fixed/changing) or availability of resources (supply/demand).
- Accounting: calculates the actual cost of the submitted request based on the use of resources. The resource used may be the processing power, file storage, network bandwidth or mix of these.
- Virtual Machine Monitor: tracks available virtual machines and allocated resources to those virtual machines.
- Dispatcher: starts the execution of accepted service requests on allocated virtual machines.
- Service Request Monitor: monitors the execution of accepted service request.
- Virtual Machines: The accepted submit requests are executed on virtual machines. Virtual machines can be started and stopped on demand. Multiple virtual machines can be run on same physical machine which provide maximum flexibility to configure various resource partitions on the same physical machine to fulfill different specific requirements of service.
- Physical Machines: The datacenter, where Cloud is hosted, is built of multiple powerful servers which provide resources for executing submitted requests.

As, discussed virtual machines play an important role in Cloud computing. Grid computing is also migrating towards virtualization which will be discussed in later. The following section introduces virtualization.

2.4 Virtualization

In 1960, IBM first developed virtual machine to provide concurrent and interactive access to a mainframe computer. Virtual machine [5] provides an elegant and transparent way to enable time-sharing and resource-sharing of high-end computing resources. Users experience an illusion that they are accessing the physical machine directly. Users could execute, develop and test applications without damaging other user experience. Virtual machines provide a fully protected and isolated instance of physical resources.

Virtualization provides a computer implemented in software within a computer. Virtualization allows running multiple virtual machines with same or heterogeneous operating systems side by side in same physical host. All the physical resources are emulated so that virtual machines can share a common set of virtual resources.

2.4.1 Benefits of Virtualization

Virtualization provides the following benefits [23] [24]:

- Consolidation to reduce hardware cost:
 - Virtualization enables efficient access and management of resources to reduce operations and systems management costs while maintaining needed capacity.
 - Virtualization enables a single server to function as multiple virtual servers.
- Optimization of workloads:
 - Virtualization enables capability of responding dynamically to the application needs of its users.
 - Virtualization can increase the use of existing resources by enabling dynamic sharing of resource pools.
- IT flexibility and responsiveness:
 - Virtualization enables a single, consolidated view and easy access to all available resources in the network, regardless of location.
 - Virtualization enables capability to reduce the environment management by providing emulation for compatibility, improved interoperability and transparency.

2.4.2 Types of Virtualization

Virtualization can be broadly categorized in three categories depending upon the resources being virtualized [25].

2.4.2.1 System Virtualization

System virtualization enables creation of many virtual machines within a single physical system.

Virtual machines are independent of the host operating system. System virtualization can be achieved by hardware partition or hypervisor technology. Physical server is subdivided into fractions using hardware partitioning. Each partition can run an operating system. Hypervisors use a thin layer of code in software or firmware to achieve fine-grained, dynamic resource sharing. Because hypervisors provide the greatest level of flexibility in how virtual resources are defined and managed, they are the primary technology of choice for system virtualization.

There are two types of hypervisor:

- Type 1 Hypervisors:

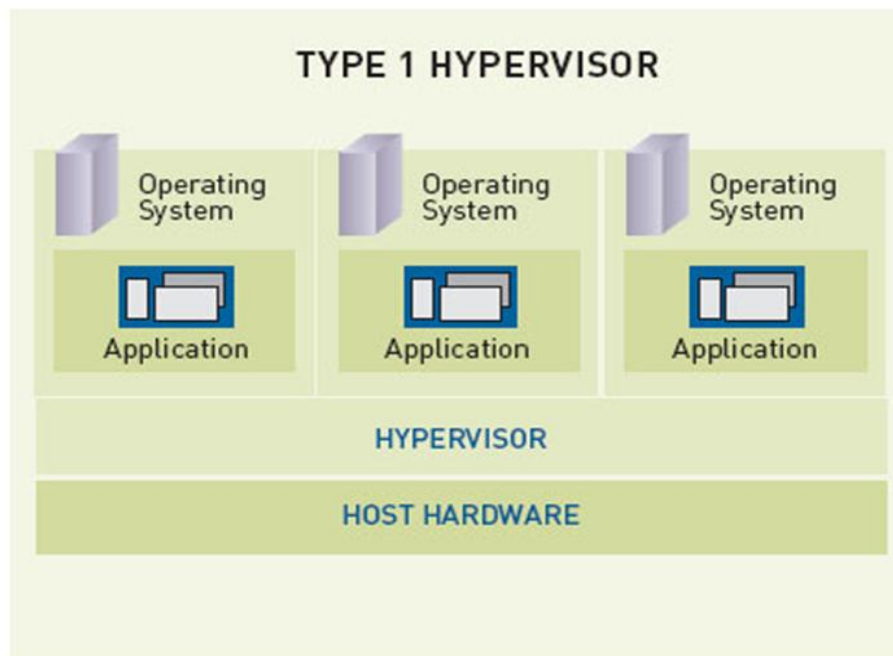


Figure 2-7 Type 1 Hypervisor.

Type 1 hypervisors run directly on the system hardware. Figure 2-7 shows one physical system with a type 1 hypervisor running directly on the system hardware, and three virtual

systems using virtual resources provided by the hypervisor. Type 1 hypervisors provide higher virtualization efficiency by dealing directly with the hardware.

- Type 2 Hypervisors:

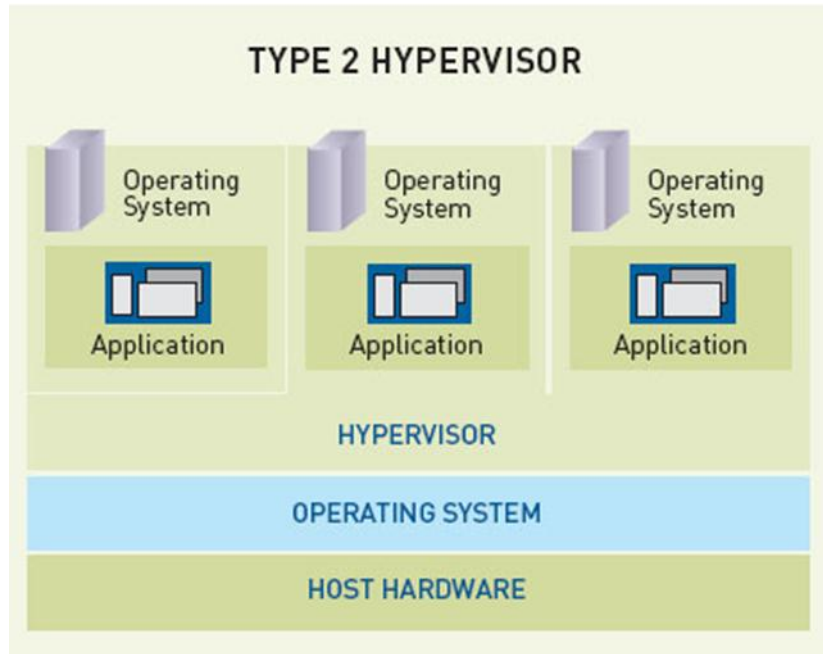


Figure 2-8 Type 2 Hypervisor.

Type 2 hypervisors run on a host operating system that provides virtualization services such as I/O device support and memory management. Figure 2-8 shows one physical system with a type 2 hypervisor running on a host operating system and three virtual systems using the virtual resources provided by the hypervisor.

2.4.2.2 Storage Virtualization:

The network storage devices [24] are virtualized as one big storage media. This abstracts the deployment complexity of the storage devices and make simpler to operate. It makes easy to aggregate and backup the data as well as executing daily operations.

2.4.2.3 Network Virtualization:

Separate resources of a network are combined using the network virtualization [24]. Bandwidths of the resources are split into channels to assign these resources as and when required. This allows each user to access all of the network resources from their computer. The resources can be files and folders on the computer, printers or hard drives etc.

2.5 Benefits of Virtualization in Grid Computing:

Grid computing provides access to many diverse and powerful resources. But it does not ensure any quality of service. Many Grid platforms do not support performance isolation. So activities on the same resource of users can influence each other in an uncontrollable way. For these reasons Grid cannot be used for reliable future use or time-critical applications.

Grid is usually used by scientific communities who need customized software environments. Any minor change in the environment can result in job failure. Software environment can be differed by the operating system, middleware, library environments and filesystem layout. Providing the entire configuration perfectly in a remote site administrated by other people is quite impossible. This elusive goal can be provided by virtual machines. The main advantages of using virtual machines in Grid computing are depicted below [26] [27]:

- Performance isolation: Virtual machines ensure performance for the user and the application. Applications executed in virtual machine do not face competition of concurrent processes. One virtual machine can be used for one user or one application.
- Customization: Virtual machines are fully customizable. It is possible to specify virtual hardware parameters as well as system software parameters. Users can create virtual machines with their customized software environment.
- Legacy system support: Virtual machine can entirely support legacy environments. Virtual hardware can support old operating system and the application.
- Administrator privileges: Many scientific jobs may need administrator privileges. It is not safe to provide Grid users the administrator privileges, as some user can misuse it and harm the execution system. But with virtualization, the administrative privileges can be provided to the users as the malicious actions remain confined to the virtual machines.
- Resource control: virtual machines can be customized dynamically at instantiation time. Memory size and disk size used by a virtual machine can be allocated dynamically. Scheduling policies can be implemented to limit the amount of resource utilized by the virtual machine at run time. Dynamic

resource control provides a way to limit the impact that a remote user may have on resources available for a local user.

2.6 Grid Virtualization Requirements

To deploy virtualization in Grid computing, three important issues need to be resolved. Those issues are [6] [26]:

2.6.1 High Performance Computing

Virtual machine abstracts the physical host machine and the host operating system. These extra layers of abstraction result in performance penalties. Grid computing provides access to high end computing resources to the users and users expect high performance. The virtualization technology used in Grid should be able to provide high performance and the overhead for virtualization should be least.

2.6.2 Grid middleware for virtual machines

Grid computing resources are heterogeneous in nature. Though virtual machines abstract the underlying physical resources, they do not solve the problem of resource heterogeneity. Virtual machines need the help of Grid middleware to be deployed in Grid. A Grid middleware should be able to stage in remote virtual machine images, configure the images for the local environment, and manage the life-cycle of virtual machines.

- **Image Staging:** user provided virtual images can be deployed in the remote host on request. Users get the flexibility to construct their own customized virtual machines.
- **Local Configuration:** images must be configured to run on a remote system environment. Virtual machines need suitable IP addresses, MAC addresses, disk space and memory to function.
- **Life-cycle management:** middleware should monitor and control the virtual machine life-cycle. It should be able to boot, run, shutdown and reboot the virtual machines. Site administrator should be able to shut down or kill virtual machines if needed.

- **Middleware Integration:** virtual machine middleware should integrate well with existing Grid middleware. It will reduce the deployment burdens and provide access to existing technology.
- **Local Resource Management System Integration:** virtual machine middleware should be well integrated with the LRMS, so that users can run both virtual machine jobs and regular jobs simultaneously. Users should be able to interact with virtual machine jobs as with regular jobs.

2.6.3 Virtual Machine Image Building

Simple tools to create or obtain virtual machine images should be provided to the users. Users can use dd command to image hard disk. Users can use virtualization software to create customized virtual machine images. Images can be also provided to the user as download.

2.7 Virtualization in Grid Computing

In this section some popular deployment of virtualization in Grid computing will be discussed.

2.7.1 Globus Virtualization Workspace

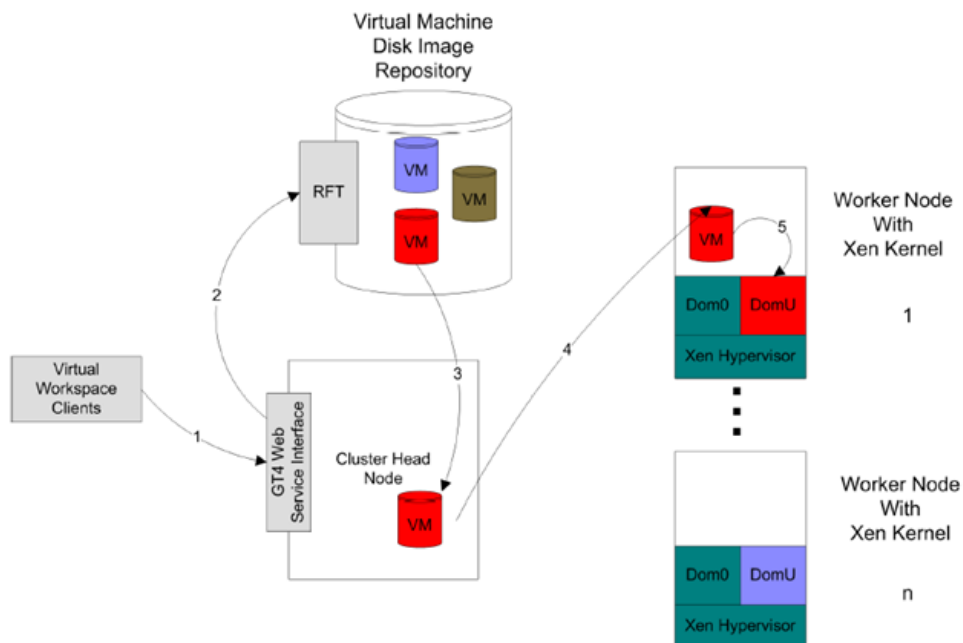


Figure 2-9 Globus Virtual Workspace [28].

Globus toolkit is the de facto middleware for Grid computing. Globus Virtual Workspaces (GVW) [28] can stage in remote VM images, configure the images for the local environment, and manage the life-cycle of the VM. GVW is well integrated with the Globus Toolkit 4 (GT4) [29]. GVW uses Web Services interface to provide an integrated mechanism for deploying and managing VMs on remote grid resources. GVW uses Xen [30] as the virtualization technology. GVW is still not integrated with a LRMS.

GVW consists of three main components. These components are discussed below:

- **Workspace Control:** resides on the cluster worker nodes. It manages the life cycle of the virtual machines by calling Xen Application Programming Interface directly.
- **Workspace Client:** installed on the user's computer. It is used to submit virtual machine jobs.
- **GT4 Web Services:** exists on the head node. It monitors the virtual machines running on worker nodes. It interacts with workspace control and workspace client using Simple Object Access Protocol (SOAP) messages.

User submits the virtual machine job using an eXtensible Markup Language (XML) file which describes the properties of the virtual machine they wish to deploy (VM name, memory requirements, hard disk size etc.) and a pointer to the remote location of the image. Figure 2-9 shows how GVW works. Virtual machine images are stored in a repository. On request, the virtual machine image is copied to the head node first, and then it is copied to the work node. In work node the workspace control controls the lifecycle of the virtual machine.

2.7.2 CernVMs

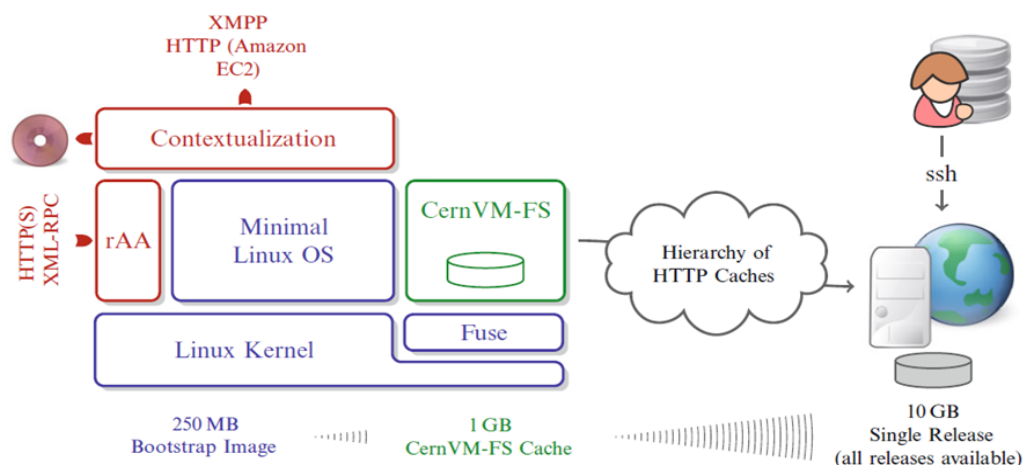


Figure 2-10 Building Blocks of CernVM [31].

CernVM [31] [28] provides complete and portable environment for developing and running LHC data analysis on any end user computer (laptop, desktop) as well as on the Grid and on Cloud resources. The primary building blocks of CernVM are shown in Figure 2-10:

- Minimal OS: contains only a minimal operating system required to bootstrap and initiate the experiment software. The minimal OS is created using rBuilder tool. rBuilder automates the OS life cycle management and the image creation process. This tool is able to generate VM images for most known virtualization technology. It also generates a change set which allows users to upgrade or downgrade their image.
- CernVM-FS: decouples the operating system from the experiment software life cycle. Pre-built and configured experiment software releases are centrally published. The releases are distributed efficiently on a large scale via a hierarchy of proxy servers or content delivery networks. The software updates are provided on repository. The updates are automatically propagated to the running virtual machine instances.
- Configuration and contextualization interfaces: configures the virtual machines to run correctly on the remote system. It mounts the software repositories before running the job. It also configures the monitoring mechanism by putting a monitoring agent inside the virtual machine and connecting it to the central monitoring system.

2.7.3 Condor VM Universe

Condor [32] is a specialized job and resource management system (RMS) which is highly used in Grid computing, for compute intensive jobs. Condor provides job management mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their jobs to Condor, and Condor subsequently chooses when and where to run them based upon a policy, monitors their progress, and ultimately informs the user upon completion.

A Condor pool consists of one central manager and multiple submit and execution nodes. Central manager is responsible for collecting information, matching available resources with resource requests. The matching of job to resource is done using ClassAds matchmaking. Every machine in Condor pool advertises their attributes to the central

manager. Jobs also have their own ClassAds which declare their requirement. Condor plays the role of matchmaker by continuously reading all job ClassAds and all machine ClassAds, matching and ranking job ads with machine ads. Condor ensures that the requirements in both ClassAds are satisfied.

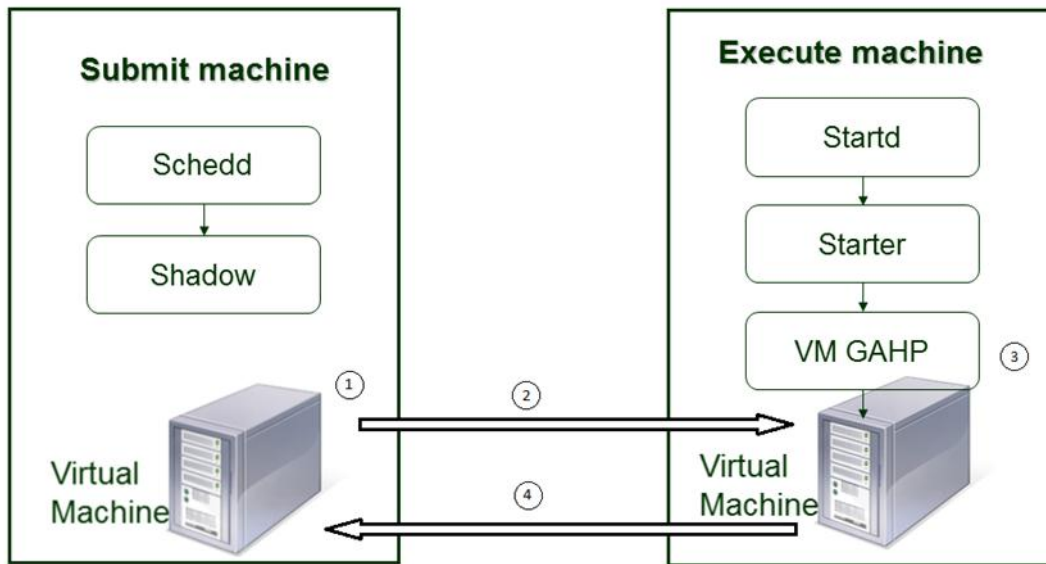


Figure 2-11 Condor VM Universe [32].

The condor vm-gahp enables the Virtual Machine Universe feature of Condor. The VM Universe uses libvirt to start and control VMs under Condor's Startd. Condor presently supports VMware server 1.0, Xen and KVM.

Users create their custom virtual machine image and submit the VM universe job with a pointer to the virtual machine image. The whole virtual machine is transferred to the execution system. On execution system, condor vm-gahp takes care of the virtual machine life cycle. It stages in the remote virtual machine, allocates memory and hard disk, configures network, and boots the virtual machines. User can interact with the virtual machine job as any other regular jobs; they can kill or put it in hold.

Users have to put the job inside virtual machine such that the job can be initiated on boot time. But condor VM universe knows nothing about what going inside the virtual machine. It cannot track the user job running inside it or it cannot recognize when the job finishes. So, users need to signal the condor about the job completion by shutting down the virtual machine on completion of the job. When the job completes, the virtual machine is transferred to the submit system.

To deploy virtualization successfully, virtual management system will be needed beforehand. The following provide a discussion about a few popular virtual machine management systems.

2.8 Virtual Machine Management

The virtualization software should be chosen depending on the kind of Virtual Machine Monitor needed, kind of operating system needed to virtualize and the underlying hardware. This section discusses about some well-known virtualization software available.

2.8.1 Xen Hypervisor

Xen [30] hypervisor runs directly on computer hardware replacing the operating system. It allows computer hardware to run multiple guest operating systems concurrently. Xen supports x86, x86-64, Itanium, Power PC, and ARM processors. Xen hypervisor can run on a wide variety of computing devices. Xen currently supports Linux, NetBSD, FreeBSD, Solaris, Windows, and other common operating systems as guests.

A computer running the Xen hypervisor contains three components:

- Xen Hypervisor: runs directly on the hardware. It is the interface where all hardware requests such as CPU, I/O, and disk are made by the guest operating systems.
- Domain 0, the Privileged Domain (Dom0): launched by the Xen hypervisor during initial system start-up. It cannot run on windows. It has unique privileges to access the Xen hypervisor. These privileges allow it to manage all aspects of Domain Guests such as starting, stopping, I/O requests etc. A system administrator can log into Dom0 and manage the entire computer system.
- Multiple DomainU, Unprivileged Domain Guests (DomU): launched and controlled by the Dom0 and independently operate on the system.

2.8.2 VirtualBox

VirtualBox [33] is a so-called "hosted" or "type 2" hypervisor. VirtualBox installed on an existing operating system and run alongside existing applications. VirtualBox is functionally identical on all of the host platforms, so virtual machine created on one host with an operating system can be used in another host with another operating system.

VirtualBox does not require the processor features built into newer hardware like Intel VT-x or AMD-V, therefore VirtualBox can be used even on older hardware where these features are not present.

VirtualBox comes with great hardware support. It supports guest multiprocessing (SMP), USB device support, full ACPI support, multiscreen resolutions, built-in iSCSI support, PXE network boot etc.

VirtualBox comes with a clean separation of client and server code with well-defined internal programming interfaces. This extremely modular design makes it easy to control it from several interfaces. Virtual machines can be controlled using graphical user interface, command line interface or API.

The VirtualBox also comes with Remote Desktop Extension (VRDE) which allows for high-performance remote access to any running virtual machine. This extension supports the Remote Desktop Protocol (RDP) originally built into Microsoft Windows, with special additions for full client USB support.

2.8.3 VMware

VMware [34] Inc. is a company who provides virtualization software. All the virtualization software is closed source and mostly commercial. They provide both server and desktop virtualization software.

VMware ESX is currently the killer software at VMware. It comes in different flavor according to the need of the user. ESXi is pretty akin to ESX except that it is free and comes with less features and hardware support.

VMware Server (formerly GSX) is major free software for server virtualization as type two hypervisor. VMware server comes in two different version, VMware server 1 and VMware server 2. VMware server 1 shipped with a rich client (VMware Server Console) to be able to connect to every compatible server. VMware Server version 2 is VMware Virtual Infrastructure compliant which is a web user interface for management.

VMware comes with a great API, VMware-vix. This high level API helps to write programs to automate virtual machine operations and run programs or manipulate files within guest operating systems. This API is easy to use, and practical for both script writers and application programmers. This API is easy to install and runs both on Linux and Windows. Bindings are provided for C, Perl, and COM (Visual Basic, VBscript, C#).

Chapter 3 Proposed Solution

This chapter discusses about the proposed solution which will enable reuse of virtual machines in Grid computing.

3.1 Proposed Solution

Currently Grid supports two methods to implement virtualization. The first method allows virtual systems to join the Grid pool as an execution system. This provides ultimate reuse of the virtual machine but as the virtual machine is configured and administrated by the pool admin, it might not possible that the virtual machine is customized for user need.

The other method allows users to submit their customized virtual machine as a job. User puts their job inside the virtual machine and configures them to initiate at system boot. User submits the virtual machine job and Grid middleware take care of virtual machine transfer. The Grid middleware configures the virtual machine to run on the remote system and power on the virtual machine. The virtual machine is configured to shutdown when all jobs completes. This shutdown signals job completion. Then the modified virtual machine is transferred to the user. This method allows user to run jobs on their customized virtual machines. But the virtual machine image has to be transferred between the user node and execute node which is very costly in terms of network utilization. Virtual machine image are huge in size and uses a lot of network bandwidth. This also limits this method to be used only in local area network.

In proposed solution, user customized virtual machines will join the pool of execution nodes. The user can submit multiple jobs using regular job submission technique. So, the problem is to enable users to power on their custom virtual machines on demand and configure the virtual machines so that it can join the execution node pool automatically.

This can be done by:

- Creating custom virtual machine system.
- Preinstalling Grid middleware or Local Resource Management system in the virtual machines.
- Configuring virtual machine to auto start the middleware daemons on powering on so it can join the pool.
- Making the virtual machine node private so no other submitter can submit jobs.

- When all jobs are completed, user must be able to shut down the virtual machine and remove it from the pool.

Virtual machines can be deployed in two ways:

3.1.1 Scenario 1

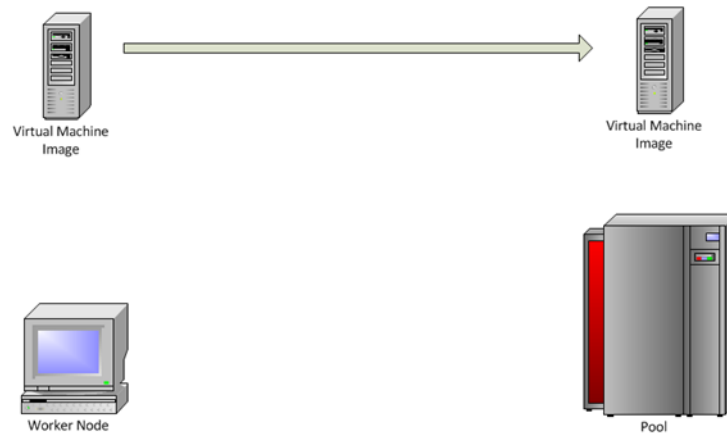


Figure 3-1 Scenario 1.

In scenario one, users create their own custom virtual machine images. The images are transferred to the execution system and powered on. Figure 3-1 shows the scenario:

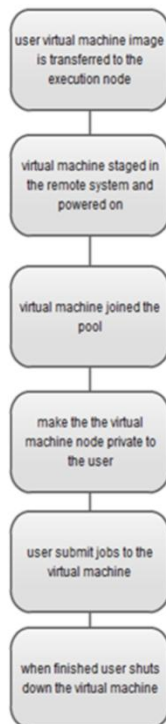


Figure 3-2 Flowchart of Scenario 1.

On remote system hard disk space, memory, processor, IP address and MAC address are allocated to the virtual machine. The virtual machine image gets registered in the virtualization hypervisor available on the remote system. The virtual machine is powered on. The Grid middleware or local resource management application are configured to be initiated on start up. When, the virtual machine starts, it joins the available resource pool. The virtual machine is made private to the user so other user can not submit jobs to that system. When all user jobs finishes, the user shuts down the virtual machine. Figure 3-2 shows the flowchart.

3.1.2 Scenario 2



Figure 3-3 Scenario 2.

A group of user usually uses same application environment for their job. So, they can create their custom virtual machine which can be pre-configured in the execution node. Like scenario 1, the virtual machine is configured to join the pool when powered on. Figure 3-3 shows the scenario.

In this scenario, virtual machine images are not transferred to the execution system. When user needs to submit jobs, user starts the virtual machine on the execution system. The Grid middleware or local resource management system runs at startup. When virtual machine joins the pool, user makes the virtual machine private for him. Then user runs his jobs. User shuts down the virtual machine when he finishes running all his jobs. Flowchart of scenario 2 is shown in figure 3-4:

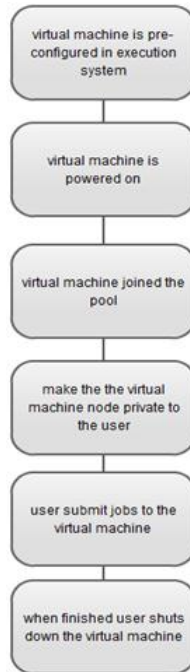


Figure 3-4 Flowchart of Scenario 2.

The proposed solution can be viewed as Cloud on Grid, as virtual machines are instantiated on user request. Next section introduces the applications used in the solution.

3.2 Application Used

To deploy the proposed solution the following applications are used:

- Condor:

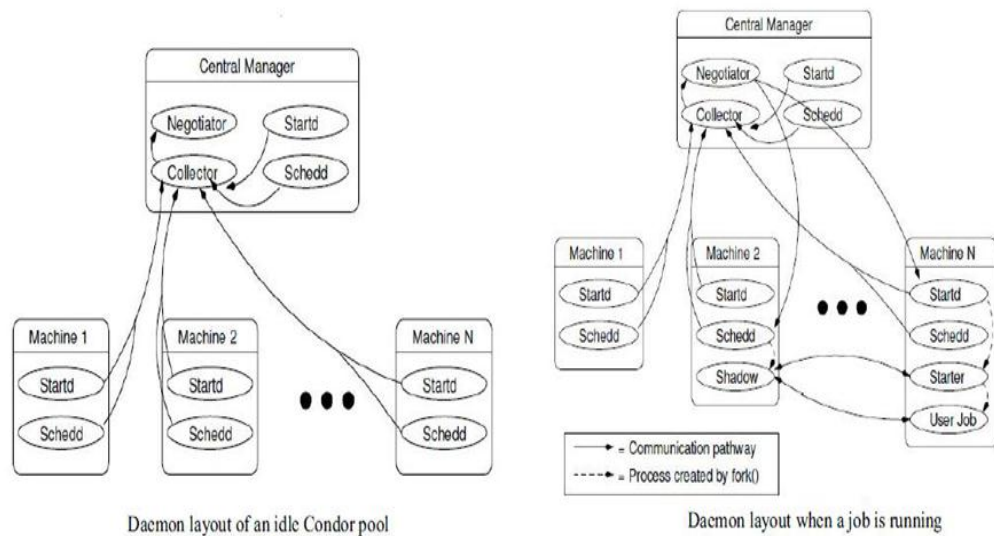


Figure 3-5 Condor Pool and Daemons [32].

Condor [32] is already introduced in the section 2.7.3. Here short descriptions of the condor daemons are provided. Figure 3-5 shows all the important daemons.

- Condor_master: runs in every node in the condor pool. This daemon administrates other daemons. It checks other daemons periodically and restarts the crashed daemons.
 - Condor_startd: runs in the execute nodes. It advertises resources attribute to the pool. It also enforces the criteria for when a job will be started, suspended, resumed, vacated, or killed. It spawns condor_starter to execute a job.
 - Condor_starter: sets up the execution environment and monitors the jobs. It also sends back the status information to the submitting machine when the job completes.
 - Condor_schedd: runs in the submit nodes. It manages the user submitted job queue. Schedd advertise the jobs. It also matches the jobs with available resources. Whenever resources become available for a job, it spawns condor_shadow to serve the job request.
 - Condor_shadow: runs on the submit nodes and acts as resource manager for submitted job request.
 - Condor_collector: runs on central manager. It collects all the information about the condor pool from all the daemons. All other daemons periodically send ClassAd updates to it.
 - Condor negotiator: runs in the central manager node. This daemon is responsible for all the match-making within the Condor system
-
- VMware Server 1: VMware Server 1 [35] is used as the virtualization application. VMware Server is a free virtualization product. It works with both Microsoft Windows and Linux systems. It allows user to partition the physical host into multiple virtual machines. It supports standard X86 hardware. Linux, NetWare, Solaris, and Windows operating systems can be installed as the guest. It experimentally supports two-way virtual SMP, Intel virtualization technology. Fedora 9 is used as the guest operating system. VMware Server 1 supports VMware-vix API [36]. This API is used in scripting powering on and powering off codes.

3.3 Deployment Details:

Deployments of the proposed scenarios differ significantly. This section gives detailed description of deployment of the both scenarios.

3.3.1 Scenario 1:

The deployment of scenario 1 requires transferring the virtual machine to the execution host. In the execution system, the virtual machine should get registered in the local virtualization application and then it should be powered on.

Condor VM universe is used to deploy this scenario. For this Condor VM universe should be configured in execution node. VMware server 1 is installed in execution node and the condor configuration file is modified to enable the VM universe.

On the submit node, the user will prepare the custom virtual machine. In the virtual machine, the user will install the Condor software to make it a condor execution node. The virtual machine is configured to automatically login with condor user when powered on. A script is created to start the Condor daemons. This script is put in the auto start programs. So, when the virtual machine will powered on, it will automatically login with condor user and starts the condor daemons. If condor is perfectly configured, the virtual machine will join the condor pool.

The virtual machine node should be private to the submitting user. This can be done by changing the machine ClassAds of the virtual machine, including client machine attribute in the start expression. When properly configured only jobs from specific submit node can be run on that execution node, jobs submitted from other submit nodes will be rejected.

The user will submit a job using condor VM universe with a pointer to the folder where the virtual machine image is stored. Condor will take care of transferring the image files to the execution node. When the file transfer completes, condor registers the virtual machine to the execution node's VMware server 1. Then, condor powers on the virtual machine. Condor uses condor VM GAHP to do all this work.

The user waits till the virtual machine execution node joins the condor pool. User submits jobs to the virtual machine execution node. This can be done by setting target machine in job requirement attribute in the job submit file.

Condor VM universe does not provide any method for shutting down the virtual machine running on the execution system. So, to shut down the virtual machine user kills the VM universe. This also clears the transferred virtual machine image files from execution node.

3.3.2 Scenario 2

In scenario 2, the virtual machine will be already available in the execution system. So, there is no need for transferring virtual machine image to the execution system and registering it. The virtual machine should be booted up on user request and join the condor pool. Like scenario 1, virtual machine is configured to automatically log in with condor user. Condor is pre-installed and configured, a script is written to start condor daemons and it is put in the startup.

VMware-VIX API is used to create code for powering on and powering off the virtual machines. To start a virtual machine it should be registered in the VMware Server 1. The path of the virtual machine configuration file location (.vmx) is provided for powering on or power off the virtual machine. The maximum number of virtual machine is predefined, to limit the number of running virtual machine, as running too much virtual machine on same host can result in system slow down or may be in crash. The power on code checks for running virtual machines. It checks for the configuration file of the running virtual machines to evaluate if another instance of the virtual machine is already running. If the number of running virtual machines is less than the number of maximum virtual machines that can run on the host and another instance of the requested virtual machine is not already running, the requested virtual machine is powered on. Else subsequent error is reported to the user. The algorithm is provided below:

```
function startVM ()
    Input: path = location of virtual machine configuration file requested to power on.
           maxVM = maximum number of virtual machine can be run concurrently
           startVM = TRUE
           runningVM = 0
    function checkVM()
        checks for running virtual machine instances
        for each running virtual machine
            loc = location of configuration file of running virtual machine
            if ( path equals loc)
                startVM = FALSE
            runningVM++
    if (runningVM < maxVM && startVM == TRUE)
        power on requested virtual machine
```

The code for powering off also checks for running virtual machines. It checks whether any virtual machine is running. It also checks the configuration file location of the running virtual machines. If it finds any running virtual machines, it compares the configuration file location to check if the virtual machine which is requested for shutting down is running or not. If there is match, then the code shuts down the virtual machine. Else error is reported to the user.

```
function stopVM ()
    Input: path = location of virtual machine configuration file requested to power off.
           stopVM = FALSE
           runningVM = 0
    function checkVM()
        checks for running virtual machine instances
        for each running virtual machine
            loc = location of configuration file of running virtual machine
            if ( path equals loc)
                stopVM = TRUE
            runningVM++
    if (runningVM > 0 && startVM == TRUE)
        power off requested virtual machine
```

The codes are compiled and the executables are stored in the execution node. User submits job to execute the power on code to start the virtual machine. When the power on job finishes, the user awaits till the virtual machine becomes available in the condor pool. Then the user accesses the virtual machine to make it private. This step is different than the scenario 1, as in scenario 1 user owns the virtual machine so he can make it private just for himself. But in scenario 2, the virtual machine is stored in the execution node and other people may use the virtual machine for submitting jobs. So, the user who starts the virtual machine should be able to make the virtual machine private to him dynamically. This can be done by setting the job start attribute to be configured on runtime. User can set new value for start on runtime and make the virtual machine private. Another user can use this value for start on runtime and make the virtual machine private. Another user can use this to make the virtual machine private to him when the first user finished his job, this enables more reuse of the virtual machine reducing overhead of shutting down and restarting it again.

When user finishes running all jobs, the user submit job to execute the power off binary on execute node. This will shuts down the virtual machine.

The next chapter discusses about the experimentation and results.

Chapter 4 Experimentations and Results

The chapter details the experimentations and results for both Scenario 1 and Scenario 2. In scenario 1 user create a customized virtual machine job and submit it as a job. In scenario 2, the virtual machine is pre-configured in execution node and it is booted on user request. The virtual machines are configured to join the existing execution node pool on powering on. So, the users can reuse the virtual machine node.

4.1 Common Configuration

The hypervisor chosen for the deployment is VMware server 1. Condor is used as the local resource manager. Condor supports virtual machine job which is discussed in section 2.7.3. This section describes the installation of condor and configuration of the custom virtual machines.

4.1.1 Condor Installation

Condor is chosen as the middleware to create the pool for the experimentation. We used Condor 7.4.4 and the pool deployed using three systems. The systems are aow4grid.uab.es, aow5grid.uab.es and aopcach.uab.es. The aow4grid.uab.es is configured as the central manager. Both aow5grid.uab.es and aopcach.uab.es serves as submit and execute node. Figure 4-1 shows the condor pool:

```
[condor@aopcach ~]$ condor_status
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@aopcach.uab.	LINUX	INTEL	Unclaimed	Idle	0.020	1762	0+00:25:04
slot2@aopcach.uab.	LINUX	INTEL	Unclaimed	Idle	0.000	1762	0+00:25:05
aow4grid.uab.es	LINUX	INTEL	Unclaimed	Idle	0.550	1518	0+00:55:04
aow5grid.uab.es	LINUX	INTEL	Unclaimed	Idle	0.820	502	0+01:33:26
Total Owner Claimed Unclaimed Matched Preempting Backfill							
INTEL/LINUX	4	0	0	4	0	0	0
Total	4	0	0	4	0	0	0

Figure 4-1 Condor Pool.

Condor user is created to install and administrate the condor middleware. Condor-7.4.4-linux-x86-rhel3-dynamic.tar.gz is downloaded from condor website. We need to create two

folders `condor-install`, where the condor will be installed, and `condor-local`, where condor stores information about executing jobs and logs.

To install condor, the tarball is extracted. Condor provides a Perl script `condor_install` to install the condor. The script needs to be executed with `--prefix=path` where the condor will be installed (`condor-install`), `--local-dir=path` of the local directory (`condor-local`) and `--type=` type of condor node. The type can be `manager` for central manager node, `submit` for job submission nodes and `execute` for job execute nodes. One pool can have just one central manager.

After installation, the condor is configured to work properly. There are two configuration file in condor. One is `condor-install/etc/condor_config` and another is `condor-local/condor_config.local`. In the `condor-install/etc/condor_config`, the `ALLOW_WRITE` and `ALLOW_READ` attributes are modified with list of nodes. The `ALLOW_READ` provides read access to the listed nodes, the listed nodes are allowed to view the status of the pool but cannot join the pool or run jobs. The nodes allowed in the `ALLOW_WRITE` can join the condor pool and submit jobs. The nodes in `ALLOW_WRITE` must also be in the `ALLOW_READ`. For the installation, the `ALLOW_READ` is set to `*` and the `ALLOW_WRITE` is set to `*.uab.es`.

The `condor-local/condor_config.local` also needs modifications. Here the condor central manager is specified using `CONDOR_HOST` parameter. For the experimental installation the `CONDOR_HOST` is set to `aow4grid.uab.es`. The network interface is specified using the `NETWORK_INTERFACE` and it is set to corresponding host IP address. By default condor waits for system inactivity for 15 minutes to start a job, but for experimentation it has been disabled by modifying the `START` attribute. The `START` attribute is set to `TRUE` so the jobs are started without any waiting.

As all the used systems hostnames and IP addresses are registered in the DNS, the `/etc/hosts` file does not have to be modified.

Condor traffic can be blocked by the default firewalls. So, the SELinux and the iptables are stopped and disabled.

Environment variables are set for easy use of condor by modifying `/etc/profile`. The `CONDOR_CONFIG` is set with the path of `condor-install/etc/condor_config`. The `condor-install/bin` and `condor-install/sbin` is included in the `PATH` environment.

The user can start condor the using the `condor_master` command. There is also a script named `condor.boot` is provided in the `condor-install/etc/example/` folder. Users need to

modify the MASTER variable with condor-install/sbin/condor_master. Users can start or stop the condor daemons by running condor.boot. Users can view the condor pool by condor_status, submit jobs by condor_submit job-submit-file, and query the submitted jobs using condor_q.

4.1.2 Virtual Machine Image Creation:

VMware server 1 is used to create the virtual machine image. Fedora 9 is chosen as the guest operating system. The bridge networking is selected as the virtual machine needs to be in the same subnet of host system to join the condor pool. The virtual machine image is made public and created using condor user. The hostname is provided manually. When the guest operating installation completes, condor user is created and condor middleware is installed.

To minimize the memory usage, the default run level is changed from X11 to full multiuser mode. This can be done by modifying /etc/inittab, changing it from

```
id:5:initdefault
```

to

```
id:3:initdefault.
```

To enable automatic login for the condor user, the /etc/event.d/tty1 is modified. It is changed to

```
exec /sbin/mingetty --autologin condor tty1
```

from the default

```
exec /sbin/mingetty tty1.
```

The condor daemons should be automatically started at the startup. The condor.boot script is modified to:

```
CONDOR_CONFIG=/home/condor/condor-install/etc/condor_config
```

```
export CONDOR_CONFIG
```

```
MASTER=/home/condor/condor-install/sbin/condor_master
```

```
if [ -x $MASTER ]; then
```

```
    echo "Starting up Condor"
```

```
    $MASTER
```

```
else
```

```
    echo "$MASTER is not executable. Skipping Condor startup."
```

```
    exit 1
```

fi

The script is created and named as condor.start. The script is turned executable. The script is included in startup by modifying /etc/rc.d/rc.local.

The virtual machine is configured with static IP address. This is done by modifying /etc/sysconfig/network-scripts/ifcfg-eth0 and allocating IP address, gateway and DNS statically. The virtual machine will use this IP configuration when booted up. Figure 4-2 shows the virtual machine system, aow12grid.uab.es, in the condor pool.

```
Every 1.0s: condor_status                               Wed Jun 29 17:08:09 2011
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@aopcach.uab.	LINUX	INTEL	Unclaimed	Idle	0.010	1762	0+02:45:04
slot2@aopcach.uab.	LINUX	INTEL	Unclaimed	Idle	0.000	1762	0+02:45:05
aow12grid.uab.es	LINUX	INTEL	Unclaimed	Idle	2.860	157	0+00:00:04
aow4grid.uab.es	LINUX	INTEL	Unclaimed	Idle	0.010	1518	0+00:31:16
aow5grid.uab.es	LINUX	INTEL	Unclaimed	Idle	1.390	502	0+00:01:26
Total Owner Claimed Unclaimed Matched Preempting Backfill							
INTEL/LINUX	5	0	0	5	0	0	0
Total	5	0	0	5	0	0	0

Figure 4-2 Condor Pool with Virtual Machine Node.

4.2 Scenario 1

To deploy scenario 1, Condor VM universe needs to be configured. The user needs to create their virtual machine image and customized it as section 4.1.2. The details are given below:

4.2.1 Condor VM Universe Configuration

To configure condor VM universe, the first step is to deploy the condor as an execution node. Then VMware Server 1 is installed with libvirt. The VM universe is configured in the aow4grid.uab.es system. The condor-install/etc/condor_config is modified to enable the condor VM universe. The attributes needed to modified are located in the VM Universe Parameters section and VM Universe Parameters specific to VMware section. These sections start at around 2180th line. The VMWARE_TYPE parameter is set to vmware and the version set using VM_VERSION = server1.0.4. The memory used by the virtual machine is specified using VM_MEMORY.

The networking is enabled by setting `VM_NETWORKING=TRUE` and specifying `VM_NETWORKING_TYPE=bridge`. The default virtual machine networking is set to bridge by `VM_NETWORKING_DEFAULT_TYPE=bridge`. The maximum number of virtual machine job can be defined by `VM_MAX_NUMBER`. The `VM_STATUS_INTERVAL`, `VM_GAHP_REQ_TIMEOUT` and `VM_RECHECK_INTERVAL` parameters are uncommented.

In VM Universe Parameters specific to VMware section, the `VMWARE_NETWORKING_TYPE` is changed to “bridged” from the default “nat”.

4.2.2 Virtual Machine Image Creation and Submitting VM Universe Jobs

The required virtual machine image is created using the methodology discussed in section 4.1.2. The fedora 9 is selected as the guest operating system. The virtual machine is created in `/var/lib/vmware/Virtual Machines/` folder.

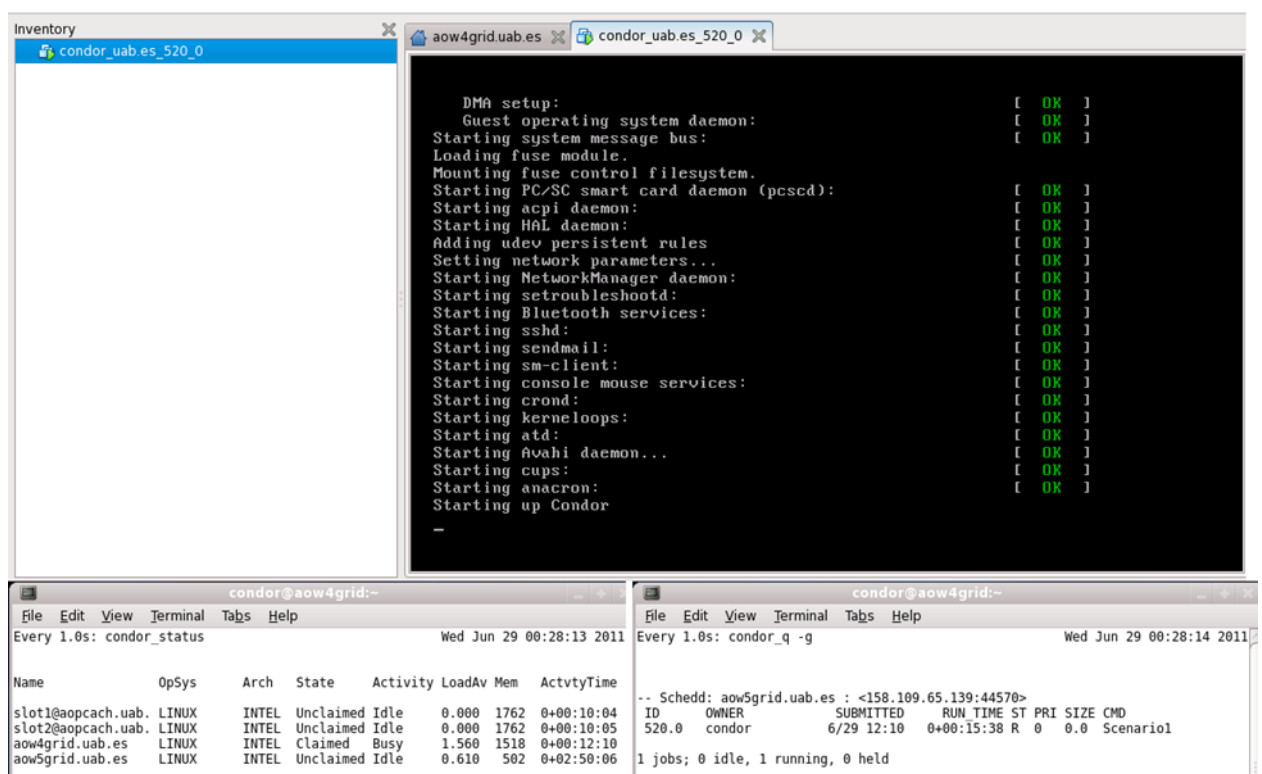


Figure 4-3 Virtual Machine Starting Up.

User creates a VM universe job submit file to submit the job. The condor job submit file is:

```

Universe           = vm
Executable         = Scenario1
Log                = Scenario1.log
vm_type            = vmware

```

```

vm_memory                = 256
vm_networking            = true
vm_networking_type       = bridge
vmware_dir               = /var/lib/vmware/Virtual Machines/Fedora/
vmware_should_transfer_files = Yes
Queue

```

The executable name is random. User points to the virtual machine image file specifying the folder using the `vmware_dir` with the location of virtual machine image file. User also specifies the memory used by the virtual machine. User defines to use the bridge type networking. Figure 4-3 shows that VM universe job has been submitted from the `aow5grid.uab.es` to `aow4grid.uab.es`. The figure also shows that the virtual machine is booting up on the remote system.

Before submitting the job, user needs to make the virtual machine execute node private so other user can not submit job to this execute node.

4.2.3 Making the Virtual Machine Private

Condor uses the `START` parameter to decide when to start execution of a job. The `START` parameter is modified as

```
START = (target.ClientMachine == $(FULL_HOSTNAME of submit node)).
```

This will evaluate the hostname of job submitter with the hostname provided on right side, if they are equal the job gets executed. If they are not equal the jobs get rejected.

To use the functionality properly, the configuration in job submitting node also needs a little modification. The following attributes should be included in condor configuration:

```
ClientMachine = "$(FULL_HOSTNAME)"
SUBMIT_ATTRS = $(SUBMIT_ATTRS), ClientMachine
```

With all these modifications, user can submit his jobs to the virtual machine by specifying `Requirements = (Machine == $(FULL_HOSTNAME of the virtual machine))` in the job submit file.

4.2.4 Submitting jobs the Scenario 1

The user uses the `aow5grid.uab.es` to submit the VM universe job. The user submits the job using the `condor_submit` command. The virtual machine image hard disk is allocated ten GB. So, it takes around fifteen minutes to transfer the virtual machine files to the `aow4grid.uab.es`. When the transfer completed, the virtual machine is powered on.

The user checks the pool using `condor_status` command. When user sees the virtual machine up in the pool, he starts submitting jobs.

The VM universe does not provide any means to power off the running virtual machines. So, when the user completes running all his jobs, the user kills the VM universe job using `condor_rm` to shut down the virtual machine. This also clears all the virtual machine image files from the `aow4grid.uab.es`. Figure 4-4 shows that five jobs have been submitted to the `aow12grid.uab.es` which is the virtual machine. It also shows the VM universe job running.

```

File Edit View Terminal Tabs Help
Every 1.0s: condor_q                                     Wed Jun 29 16:30:21 2011

-- Submitter: aow5grid.uab.es : <158.109.65.139:46518> : aow5grid.uab.es
ID      OWNER      SUBMITTED      RUN TIME ST PRI  SIZE CMD
529.0   condor     6/29 16:12    0+00:17:39 R  0   0.0 Scenario1
530.0   condor     6/29 16:30    0+00:00:02 R  0   0.0 simple 4 10
530.1   condor     6/29 16:30    0+00:00:00 I  0   0.0 simple 4 10
530.2   condor     6/29 16:30    0+00:00:00 I  0   0.0 simple 4 10
530.3   condor     6/29 16:30    0+00:00:00 I  0   0.0 simple 4 10
530.4   condor     6/29 16:30    0+00:00:00 I  0   0.0 simple 4 10

6 jobs; 4 idle, 2 running, 0 held

File Edit View Terminal Tabs Help
Every 1.0s: condor_status                               Wed Jun 29 16:30:29 2011

Name                OpSys      Arch  State      Activity LoadAv Mem  ActvtyTime
slot1@aopcach.uab. LINUX      INTEL Unclaimed Idle      0.000 1762 0+02:10:04
slot2@aopcach.uab. LINUX      INTEL Unclaimed Idle      0.000 1762 0+02:10:05
aow12grid.uab.es    LINUX      INTEL Claimed  Busy      0.540 248  0+00:00:02
aow4grid.uab.es     LINUX      INTEL Claimed  Busy      1.520 1518 0+00:12:52
aow5grid.uab.es     LINUX      INTEL Unclaimed Idle      0.160 502  0+02:10:05
Total Owner Claimed Unclaimed Matched Preempting Backfill
INTEL/LINUX        5      0      2      3      0      0      0
Total              5      0      2      3      0      0      0

```

Figure 4-4 Job Submission in Scenario 1.

4.3 Scenario 2

The same virtual machine is used in Scenario 2. The execution system is the `aow5grid.uab.es`. The user uses the `aopcach.uab.es` system.

4.3.1 Configuring Execution Node:

In scenario 2, the virtual machines are preconfigured in execution node. The virtual machine is powered on and powered off on user's request. The VMware server 1 is installed in the execution node. The VMware-VIX has to be updated to the current version. The code for powering on and powering off is written as discussed in the section 3.3.2. The codes are compiled and the full paths of the executables are provided to the users.

In the execution node, the virtual machine is created as described in the section 4.1.2.

```

File Edit View Search Terminal Help
Every 1.0s: condor_q                               Wed Jun 29 17:05:50 2011

-- Submitter: aopcach.uab.es : <158.109.65.21:45242> : aopcach.uab.es
ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
 33.0   condor      6/29 17:04    0+00:00:25 R  0   0.0  startVM

1 jobs; 0 idle, 1 running, 0 held
File Edit View Search Terminal Help
Every 1.0s: condor_status                          Wed Jun 29 17:05:49 2011

Name                OpSys      Arch   State   Activity LoadAv Mem   ActvtyTime
slot1@aopcach.uab. LINUX      INTEL  Unclaimed Idle    0.100 1762 0+02:40:04
slot2@aopcach.uab. LINUX      INTEL  Unclaimed Idle    0.000 1762 0+02:40:05
aow4grid.uab.es     LINUX      INTEL  Unclaimed Idle    0.000 1518 0+00:26:16
aow5grid.uab.es     LINUX      INTEL  Claimed  Busy    0.000  502 0+00:00:04

```

Figure 4-5 Powering On the Virtual Machine.

4.3.2 Making the Virtual Machine node Private:

In scenario 1, the user has the ultimate control on the virtual machine. But in scenario 2, the virtual machine is used by group of users. So, a dynamic method is required to make the virtual machine node private. This can be done by enabling dynamic reconfiguring by setting

`ENABLE_RUNTIME_CONFIG = TRUE`

in local configuration file. The `START` parameter will be modified to make the virtual machine private. The dynamic modification of `START` is enabled by

`SETTABLE_ATTRS_CONFIG = START.`

The attribute is modified on runtime by

```
condor_config_val -startd -rset "START = (target.ClientMachine == $(FULL_HOSTNAME of submit node))"
```

and then running

```
condor_reconfig --startd
```

to apply the modified attribute.

4.3.3 Powering on the virtual machine:

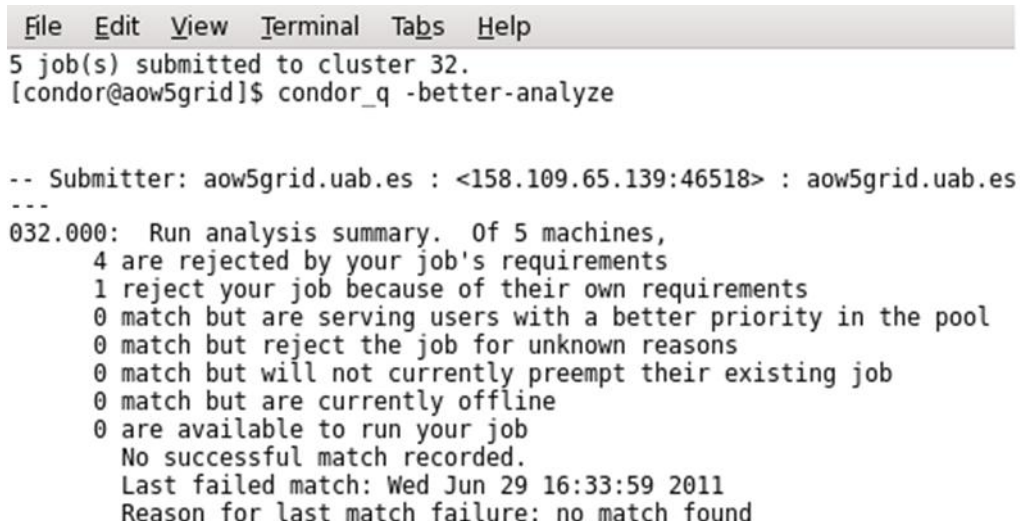
The user submits a vanilla universe job to start the virtual machine. The job submission is shown in Figure 4-5.

In the job submit file, transfer executable is set to false as the executable is in execution system. The executable file path is given using the Executable parameter. The Requirement parameter is used to define the executable system. The job is submitted by condor_submit.

4.3.4 Submitting job

If the job succeeds and there is no error message in the output file, the user waits till the virtual machine becomes available in the condor pool. User enquires the condor pool using the condor_status command. When it becomes live, user login to virtual machine using secure shell and make the virtual machine private as discussed in section 4.3.2.

Figure 4-6 shows errors obtained when other users try to submit jobs to virtual machine. The virtual machine aow12grid.uab.es is private to the aopcach.uab.es. If user from aow5grid.uab.es tries to submit job to the virtual machine, the jobs are rejected by the virtual machine.



```
File Edit View Terminal Tabs Help
5 job(s) submitted to cluster 32.
[condor@aow5grid]$ condor_q -better-analyze

-- Submitter: aow5grid.uab.es : <158.109.65.139:46518> : aow5grid.uab.es
---
032.000: Run analysis summary. Of 5 machines,
  4 are rejected by your job's requirements
  1 reject your job because of their own requirements
  0 match but are serving users with a better priority in the pool
  0 match but reject the job for unknown reasons
  0 match but will not currently preempt their existing job
  0 match but are currently offline
  0 are available to run your job
  No successful match recorded.
  Last failed match: Wed Jun 29 16:33:59 2011
  Reason for last match failure: no match found
```

Figure 4-6 Job Submission Errors.

Then user submits his jobs to the virtual machine node specifying the execution node using Requirement parameter.

4.3.5 Powering off the Virtual machine:

When user finishes running all the jobs, user submit another job to shut down the virtual machine. The job submit file is similar to the job submit file used for powering on. The Executable parameter is changed to point to the executable for powering off.

When the virtual machine shuts off, the condor pool is not updated instantly. As condor collector needs some time interval to update its status.

4.4 Evaluation of results

The scenarios are deployed in the experimental condor pool. The results are evaluated based on:

- Powering on the virtual machine: the scenario1 gives more transparency than the scenario 2. In scenario 1 user just need to submit a condor VM Universe job. But in scenario 2 user needs to know about the location of VIX scripts. The user also needs to know which execution node has the virtual machine installed.
- Time to become live on pool: In scenario 1 virtual machine takes fifteen minutes to become live on pool, where in scenario 2 it takes only two minutes. In scenario 2, the virtual machine is preconfigured. So, the virtual machine boots up and joins the pool. But in scenario 1, the virtual machine images are in user system. The image needs to transfer to the execution node. The size of virtual machine image used for the experimentation is ten gigabytes. The transfer takes around fifteen minutes.
- Submitting jobs: in scenario 1, the user is the administrator of the virtual machine. No other user can access or modify the virtual machine. The virtual machine is made private before submitting the virtual machine job. So, when the machine becomes live on pool, the user just can start submitting his jobs. But in scenario 2, the virtual machine image is shared between groups of users. The user needs to login to the remote virtual machine and change the START attribute to make it private.
- Shutting down the virtual machine: user just needs to kill the running VM universe job to power off the virtual machine. But in scenario 2, user needs to submit another job using condor to shutdown the virtual machine.
- Fault tolerance: in the scenario 1, the virtual machine process runs as a condor job. So, condor can migrate or suspend the virtual machine job in case heavy load or

memory shortage. Condor can checkpoint the VM universe jobs. But in scenario 2, the virtual machine runs independent of condor. The virtual machine acts as a separate entity. The host system and the virtual machine do not know about each other's work load or memory usage. So in case of high work load or memory usage, there is no way to provide fault-tolerance.

- Network usage: the scenario 2 uses the normal network bandwidth. But scenario 1 generates huge network traffic due to transfer of virtual machine image. It is also possible that some malicious user can start a denial of service attack submitting a lot of VM universe job with huge virtual machine images. Figure 4-7 shows network bandwidth usage.

	19.1MB	38.1MB	57.2MB	76.3MB	95.4MB			
aow5grid.uab.es => aow4grid.uab.es			85.1MB	80.6MB	83.0MB			
			<=	1.50MB	1.44MB			
aow5grid.uab.es => aopcach.uab.es			6.99KB	23.0KB	13.3KB			
			<=	17.0KB	8.19KB			
aow5grid.uab.es => smtp-in2.uab.es			0b	2.47KB	632b			
			<=	0b	4.64KB			
aow5grid.uab.es => smtp-in.uab.es			628b	2.37KB	508b			
			<=	988b	4.64KB			
aow5grid.uab.es => dns.uab.es			1.41KB	1.41KB	462b			
			<=	5.40KB	5.05KB			
aow5grid.uab.es => xpv-uab-cat.uab.es			2.97KB	2.42KB	3.58KB			
			<=	160b	160b			
158.109.255.255 => etse-70-67.uab.es			0b	0b	0b			
			<=	2.71KB	555b			
255.255.255.255 => ee-68-229.uab.es			0b	0b	0b			
			<=	0b	267b			
255.255.255.255 => etse-75-70.uab.es			0b	0b	0b			
			<=	0b	217b			
<hr/>								
TX:	cumm:	5.07GB	peak:	85.9MB	rates:	85.2MB	80.6MB	83.0MB
RX:		91.5MB		1.57MB		1.53MB	1.47MB	1.47MB
TOTAL:		5.16GB		87.3MB		86.7MB	82.1MB	84.5MB

Figure 4-7 Network Bandwidth Usage.

The figure shows that the VM universe job uses 80Mbps network bandwidth on average where a normal test vanilla job uses only 10Kbps network bandwidth on average.

The next chapter concludes this thesis. It also discusses some future work which can improve the deployment.

Chapter 5 Conclusions and Future Work

The virtualization can be used in Grid computer to provide customized and specific job execution environment to the users. Cloud computing provides its services using virtualization. In this master thesis, reuse of virtual machines has been explored using Condor and VMware server 1 virtualization application.

5.1 Conclusions

Grid Computing can provide access to high end computing resources but it cannot guarantee quality of service. Many Grid middleware does not support performance isolation. Grid computing also cannot provide specific environment configuration for complex scientific applications.

Virtualization can be used to overcome these limitations. Cloud computing has successfully implemented virtualization to provide services to its users. Cloud computing supports on demand provisioning of hardware using virtualization and provides an illusion of infinite resources.

Grid computing can incorporate virtualization in two ways. The user can create their customized virtual machine. Then user puts his jobs at the startup. The user has to configure the virtual machine to shutdown on job completion. The user submits the whole virtual machine as a job. The Grid middleware transfers the virtual machine to the execution node, configures the virtual machine for the local environment. The middleware also powers on the virtual machine. The virtual machine boots and starts executing the jobs. The virtual machine shuts down on all job completion, which signals the middleware that the virtual machine job is completed. The middleware then transfers the modified virtual machine image to the user. The second way is to use virtual machine as execution node. The middleware is installed in the virtual machine so that they can join the pool. The virtual machines node act like another normal node in the pool.

The first method gives the user flexibility for customization and the second methods gives the reusability. The proposed solution combines both methods to provide both customization and reusability. In proposed solution user installs middleware in the custom virtual machines. User configures the middleware to initiate on system startup. Then the

user submits the virtual machine as a job. When the virtual machine is powered on in the remote execution, it joins the pool. So, the user can submit his jobs to the virtual machine. The user also needs to make the virtual machine private so that other user cannot submit job to his virtual machine.

The proposed solution can be deployed another way. The mentioned deployment policy is called in this master thesis as scenario 1. In scenario 2, the custom virtual machine is preconfigured in the execution node. The virtual machine gets powered on and powered off on user request. Like the scenario 1, the virtual machine joins the pool.

The scenario 1 can be deployed using the condor VM universe. Condor transfers the files to the execution node and powers on the virtual machines. User submits his jobs when the system becomes live on pool. User kills the VM universe job to shutdown the virtual machine.

This scenario needs to transfer the virtual machine image files to the execution time. This transfer may take several minutes. Except this bottleneck this scenario provides complete power to the user. As the administrator of the virtual machine, it is easy for the user to make it private. Other user cannot access or modify the system.

In scenario 2, the virtual machine on remote execute system needs to be powered on or powered off on user request. This is done by scripting separate codes for power on and power off the virtual machine. The codes are run by submitting separate condor jobs.

This scenario needs high user mutual understanding. The virtual machine image is shared between users. Any user can power on or power off the virtual machine at any time. So it is necessary that users must respect other users and do not manipulate the virtual machine when another user is using it. Also the virtual machine runs completely separated from the host system. They don't know about each other's load or memory usage. This can be resulted in system crash.

5.2 Future Work

The scenario 1 and scenario 2 will be tested using CloudSim to see its impact on large cluster.

For the scenario 1, the condor VM universe is used. To improve the scenario1, condor VM universe needs to be improved.

- Configuring condor to allow static NIC hardware address allocation. Condor creates new configuration file allocating new NIC hardware address to virtual

machines. Even if user tries to enforce static hardware address, Condor does not respect it and always change the address.

For the scenario 2:

- Communication between the host and guest system so they know about each other's load, memory usage and other statistics.
- Controlling the virtual machine usage through a web portal so no user can interfere with other user's virtual machine.
 - The portal will power on and power off the virtual machine transparently on user request.
 - The action to make the virtual machine private will be done by the portal.
 - Once one virtual machine is allocated to a user, other user cannot interact with that virtual machine image.

Bibliography

- [1] Ian Foster and Carl Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure.*: Morgan Kaufmann Publishers Inc., 2003.
- [2] Ian Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, vol. 55, 2002.
- [3] Michael Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," 2009.
- [4] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, pp. 599-616, 2009.
- [5] Sean Campbell and Michael Jeronimo, *Applied Virtualization Technology: Usage Models for IT Professionals and Software Developers.*: Intel Press.
- [6] Renato J. O., Peter A. Dinda, and José A. B., "A Case For Grid Computing On Virtual Machine," , 2003, pp. 550-559.
- [7] Jennifer M. Schopf and Bill Nitzberg, "Grids: The top ten questions," *Scientific Programming*, vol. 10, no. 2, pp. 103-111, 2002.
- [8] Rajkumar Buyya and Srikumar Venugopal, "A Gentle Introduction to Grid Computing and Technologies," *CSI Communications*, vol. 29, no. 1, pp. 9-19, 2005, Computer Society of India (CSI).
- [9] Joseph D. Sloan, *High performance Linux clusters - with OSCAR, Rocks, openMosix, and MPI.*, 2005.
- [10] Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya, "Visual Modeler for Grid Modeling and Simulation (GridSim) Toolkit," , 2003, pp. 1123-1132.
- [11] Ian T. Foster, Carl Kesselman, and Steven Tuecke, "The Anatomy of the Grid - Enabling Scalable Virtual Organizations," *CoRR*, vol. cs.AR/0103025, 2001.
- [12] I. Foster, "What is the grid? a three point checklist," 2002.
- [13] Gregor Von Laszewski, "Grid Computing: Enabling a Vision for Collaborative Research," , 2002, pp. 37-52.
- [14] Fran Berman, Geoffrey Fox, and Tony He, "The Grid: past, present, future".
- [15] David De Roure, Mark A. Baker, Nicholas R. Jennings, and Nigel R. Shadbolt, "The Evolution of the Grid," *Concurrency and Computation: Practice and Experience*, 2003.
- [16] Ian T. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *J. Comput. Sci. Technol.*, vol. 21, no. 4, pp. 513-520, 2006.
- [17] Types of Grid. [Online]. <http://www.it-tude.com/types-of-grids.html>
- [18] M. Haynos J., A visual tour of Open Grid Services Architecture, Accessed July 2, 2011.
- [19] Luis M. Vaquero and Luis Rodero-Merino and Juan Caceres and Maik Lindner, "A Break in the Clouds: Towards a Cloud Definition," *Computer Communication Review*, May 2010.
- [20] Ismael Chang Ghalimi. (2010, May) Intalio. [Online]. <http://www.intalio.com/benefits-of-cloud-computing>

- [21] William Voorsluys, James and Rajkumar Buyya, , A.Goscinski (eds) R., Ed.: Wiley Press, New York, USA, February 2011, ch. Introduction to Cloud Computing, pp. 1-41.
- [22] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," *Computing Research Repository*, vol. abs/0808.3, pp. 5-13, 2008.
- [23] R. Goldberg, "Survey of virtual machine research," 1974.
- [24] Susanta Nanda and Tzi cker Chiueh, "A Survey on Virtualization Technologies".
- [25] IBM Systems Virtualization. [Online].
publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf
- [26] Katarzyna Keahey, Ian T. Foster, Timothy Freeman, and Xuehai Zhang, "Virtual workspaces: Achieving quality of service and quality of life in the Grid," *Scientific Programming*, vol. 13, pp. 265-275, 2005.
- [27] Lizhe Wang, Gregor Von Laszewski, Marcel Kunze, Jie Tao, and Jai Dayal, "Provide Virtual Distributed Environments for Grid computing on demand," *Advances in Engineering Software*, vol. 41, pp. 213-219, 2010.
- [28] A. Agarwal et al., "Deploying HEP applications using Xen and Globus Virtual Workspaces," *Journal of Physics: Conference Series*, vol. 119, 2008.
- [29] Globus Toolkit. [Online]. www.globus.org/toolkit/
- [30] Paul Barham et al., "Xen and the art of virtualization," , 2003, pp. 164-177.
- [31] P. Buncic, C. Aguado Sánchez, J. Blomer, A. Harutyunyan, and M. Mudrinic, "A practical approach to virtualization in HEP," *The European Physical Journal Plus*, vol. 126, pp. 1-8, 2011, 10.1140/epjp/i2011-11013-1.
- [32] Condor Project Homepage. [Online]. <http://www.cs.wisc.edu/condor/>
- [33] VirtualBox. [Online]. <http://www.virtualbox.org/>
- [34] VMware. [Online]. <http://www.vmware.com/>
- [35] VMware server. [Online].
http://downloads.vmware.com/d/info/datacenter_downloads/vmware_server/1_0
- [36] VIX API. [Online]. <http://www.vmware.com/support/developer/vix-api/>

