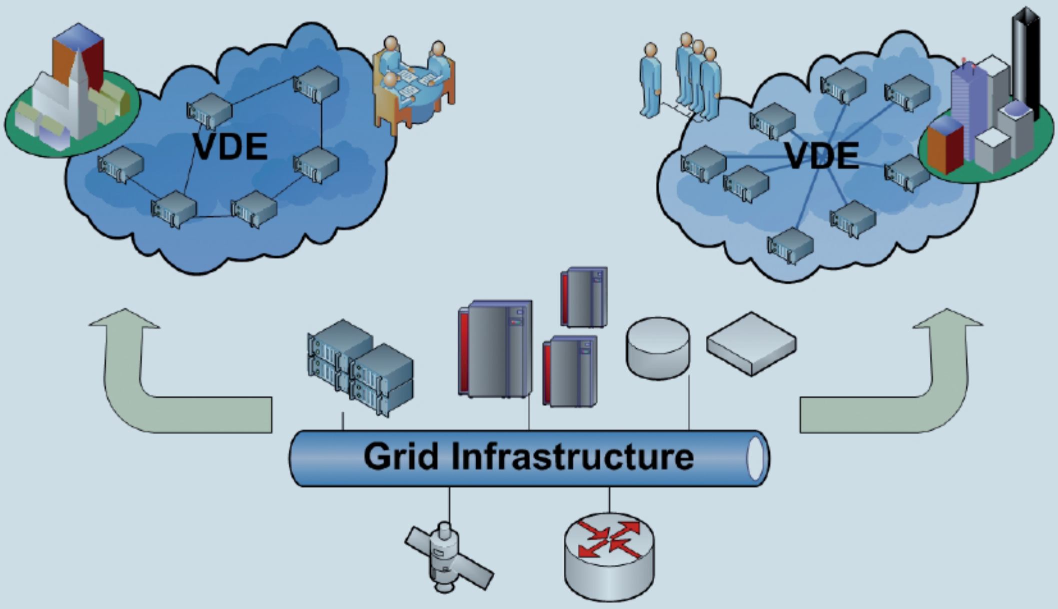
Schriftenreihe des Instituts für
Angewandte Informatik / Automatisierungstechnik
Universität Karlsruhe (TH)
Band 25

Lizhe Wang

Virtual Environments for Grid Computing



universitätsverlag karlsruhe

Lizhe Wang

Virtual Environments for Grid Computing

Schriftenreihe des

Instituts für Angewandte Informatik / Automatisierungstechnik

an der Universität Karlsruhe (TH)

Band 25

Virtual Environments for Grid Computing

von
Lizhe Wang

universitätsverlag karlsruhe

Dissertation, Universität Karlsruhe (TH)
Fakultät für Maschinenbau, 2008

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe
www.uvka.de

Universitätsverlag Karlsruhe 2008
Print on Demand

ISSN: 1614-5267
ISBN: 978-3-86644-310-5

Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Virtual Environments for Grid Computing

Submitted as Doctor Dissertation
in partial fulfillment of the requirements for the Doctor of Engineering

Department of Mechanical Engineering
University Karlsruhe (TH)

by
Master of Engineering Lizhe Wang

Advisor:	Prof. Dr. Ing. habil. Georg Bretthauer
Co-advisor:	Dr. rer. nat. habil. Marcel Kunze

Date of submission:	11. November 2008
Date of oral defence:	17. December 2008

Zwei Dinge erfüllen das Gemüt mit immer neuer und zunehmender Bewunderung und Ehrfurcht, je öfter und anhaltender sich das Nachdenken damit beschäftigt:der bestirnte Himmel über mir und das moralische Gesetz in mir.

—— Immanuel Kant

有两种伟大的事物，我们越是执着地思考它们，心灵就越是充满永远新鲜、有增无已的赞叹与敬畏：那就是我们头上灿烂的星空和我们内心的道德法则。

—— 伊曼奴埃尔 康德

Speziell für meine Eltern, Brüder und meine Frau.

献给一直理解、支持我的父母、兄弟和妻子

Zusammenfassung

Das Computing-Grid hat eine vollständig verteilte, dynamisch konfigurierbare und skalierbare Infrastruktur. Das Grid-Compting hat als Ziel eine flexible, sichere und koordinierte Aufteilung einer variablen Menge von Menschen, Institutionen und anderen Ressourcen. Ungeachtet dessen, dass bereits zahlreiche Fortschritte erzielt worden sind, gibt es einige Herausforderungen, die sich nicht mit den Eigenschaften des bisherigen Grid-Computings lösen lassen, wie z. B. die Bereitstellung gewünschter Benutzerumgebungen und die Qualitätssicherung.

Die Virtualisierungstechnologie ist dabei, sich zu einer vielversprechenden Lösung zu entwickeln, um erstrebenswerte kontrollierbare und flexible Informationsstrukturen aufzubauen. Es haben sich verschiedene Vorteile der Virtualisierung herausgestellt z. B. die Performance-Garantie und die Anpassbarkeit der Rechenumgebung.

Diese Doktorarbeit zeigt daher eine neue Philosophie der Grid-Nutzung auf: Das Bereitstellen von virtuellen Rechenumgebungen für Grid-Benutzer. Die neue Methodik könnte jene Probleme lösen, die gegenwärtig beim Grid-Computing auftreten. Sie unterstützt gewünschte Rechenumgebungen, welche zugeschnittene Hardware- und Softwarekonfiguration, Qualitätssicherung und steuerbare Funktionalitäten zur Verfgung stellen. Diese Philosophie befreit des Weiteren Ressourcenanbieter von verschiedenen schweren Aufgaben des Ressourcenmanagements.

Eine schlanke Middleware, Grid Virtualization Engine, die Funktionalitäten bereitstellt, um virtuelle Umgebungen in der Grid-Infrastruktur zu bauen, wird in dieser Doktorarbeit entwickelt und präsentiert. Die Arbeit diskutiert drei typische virtuelle Umgebungen, um diese Philosophie zu stützen: Grid-Workflow in einer virtuellen Box, virtuelle e-Science-Infrastruktur und virtuelles Computercenter.

Diese Doktorarbeit entwickelt die Philosophie einer Grid-Vision und Grid-Nutzung durch die Ausarbeitung einer neuen Methodik der Grid-Nutzung: Bereitstellung und Betrieb von virtuellen Umgebungen in einer gemeinsamen Grid-Infrastruktur.

Abstract

Computational Grid is a fully distributed, dynamically reconfigurable and scalable infrastructure. Grid computing aims to support flexible, secure, coordinated resource sharing among dynamic groups of individuals, institutions and resources. Despite the fact that numerous progress has been made, some challenges still obstruct the advance of Grid computing, for example, provision of desired user computing environments and Quality of Service (QoS).

Virtualization technology recently emerged as a promising solution for building desirable, manageable and flexible information infrastructures. Various advantages of virtualization have been identified, for instance, performance guarantees and computing environment customization.

This thesis therefore proposes a new philosophy of Grid usage: provide virtual computing environments for Grid users. The new methodology could solve the problems currently encountered by Grid computing. It supports desired computing environments for users, which provide customized hardware/software configuration, QoS assurance and manageable functionality. This philosophy also liberates resource providers from the various onerous tasks of resource management.

A lightweight middleware – Grid Virtualization Engine, which renders the functionality to enable virtual environments to be built on Grid infrastructures, has been developed in the doctoral work and presented in this thesis. This thesis also discusses three typical virtual environments to justify the philosophy: Grid workflow in a Virtual Box, virtual e-Science infrastructure and virtual computing center.

This doctoral work advances the Grid vision and Grid usage philosophy by rendering a new methodology for Grid usage: provision of and operation on virtual environments on shared Grid infrastructures.

Acknowledgement

I would like to express my gratitude and respect to the following people, who have contributed so much to my research recent years.

I would like to extend special thanks to Prof. Dr. Georg Bretthauer, who arranged my doctoral project in the context of the Institute of Applied Computer Science & Automation, University Karlsruhe, the top elite German University.

I also want to express my deepest gratitude to Dr. Kunze, who managed my doctoral project in the Steinbuch Center for Computing of the Karlsruhe Institute of Technology, one of six "centers of excellence" in the world.

I owe too much to my advisors, Prof. Dr. Georg Bretthauer and Dr. Marcel Kunze, for their remarkable guidance, heuristic advice and encouragement during my research. In the doctoral work, my advisors offer constant encourage in exploring novel research issues and facing new challenges. Their extensive knowledge, visionary insight and amiable personalities help so much in my research.

Dr. Jie Tao has lent many excellent suggestions for my research and helped refine my doctoral thesis. Mr. Emeric Kwemou assisted in implementing of the Grid Virtualization Service. Mr. Alvaro Canales Castellanos and Mr. Dharminder Rattu helped develop the Cumulus project. Dr. Hans Wenzel, from Fermi National Accelerator Laboratory (U.S.), offered the CMS benchmark. Mr. Bo Liu, from Nanyang Technological University (Singapore), provided the satellite image processing application.

A great debt of gratitude also goes to my colleagues:

Mr. Michael Roth, Dr. Jens-Michael Milke, Mr. Thomas Brandel, Dr. Shutao Li, Dr. Yixiang Gan, Mr. Jianfeng Xu, Mr. Fabian Kulla, Dr. Olaf Schneider, Dr. Rüdiger Berlich, Dr. Christopher Jung, Dr. Volker Büge, Mr. Oliver Oberst, Miss. Sabine Schumacher (Research Center Karlsruhe, Germany),

Prof. Dr. Wolfgang Karl, Mr. David Kramer (University Karlsruhe, Germany),

Prof. Dr. Gregor von Laszewski (Rochester Institute of Technology, U.S.), and

Dr. Wei Jie (National Center for e-Social Science, U.K.), and

Prof. Dr. Wentong Cai (Nanyang Technological University, Singapore).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation for the research

1.1.1 A retrospective look at Grid computing

Computer architectures, networks, computing paradigms and applications evolve together and are perpetually bound by mutual association. Whatever the performance of a single processor or a computer at given time, higher performance can, in principle, be achieved by utilizing many processors or distributed computing systems [18]. The last two decades represent a particularly interesting snapshot. Parallel & distributed computing take hold in many areas of mainstream computing, considering the fact that they are endorsed by dramatic technological realignments, such as rapid advantage of single-chip microprocessor, high performance networking and capacity for ever greater amounts of data storage:

- New trends in computer applications
  Demands of high performance applications are the driving force behind the progress of computing techniques.

  Some innovative applications emerge:

  - high performance distributed applications, such as modeling global climate change over long periods [26], the evolution of galaxies [123], the atomic structure of materials [5], and
  - huge data processing tasks, e.g., LHC Grid project [95] produces roughly 15 Petabytes of data annually, which are accessed and analyzed by thousands of scientists worldwide.

- Endorsement of recent computing technology achievements
  Recent computing techniques realize surprising advantages:

- rapid development of VLSI technology allows the single-chip microprocessor to become more powerful,
- high performance networks enable high bandwidth and low latency communications, and
- storage capacity for vast data sets.

These significant developments thus make it possible to build high performance distributed computing systems.

- Towards industry standards: computing paradigms
  Various industry standards in computer systems and computing paradigms are achieved, for instance, Internet protocols, distributed computing paradigms, such as, RPC/RMI, FTP/HTTP, CORBA, and Web service.

In essence we could declare that we are facing an exciting epoch: geographically distributed computing resources and storage resources, connected by high speed networking, provide a wide-area computing environment for innovative applications. Several terms have been identified for such a computing paradigm, for example, Metacomputing [112], high performance distributed computing (HPDC) [50], Global computing [33], Internet computing [71], Wide-area computing [47], and Grid computing [42][40]. This thesis adopts the term of **Grid computing**.

1.1.2 Advances of Grid computing

Grid computing has proven to be one of the most innovative components of computing techniques in the last ten years. Distinguished from conventional parallel and distributed computing, Grid computing mainly focuses on resource sharing among geographically distributed sites and the development of innovative, high-performance oriented applications. Computational Grids can present Grid users with pervasive and inexpensive access to a wide variety of resources [42].

Some examples of resources that can be shared in the Grid are listed in Table 1.1.

Table 1.1: Examples of Grid resources

Resource type	**Example**
Computing resource	Supercomputer, NOW[1], cluster
Storage resource	DPSS[2], RAID[3]
Scientific instruments	Telescope,Video equipment
Network	Dedicated link, High-performance network

Nowadays, there are possibly tens of thousands of projects and test beds involved in Grid computing. Some examples are listed below and a large number of applications have benefitted from the creation of computational Grid (see Table 1.2):

- EGEE
  Funded by the European Commission, the EGEE (Enabling Grids for E-sciencE) project [35] contains 12 partner federations, 70 contractors and 30 non-contacting participants, covering a wide-range of both scientific and industrial applications.

- OSG
  The Open Science Grid (OSG) [154] is a collaboration of science researchers, software developers and computing, storage and network providers. OSG members come from universities, national laboratories and computing centers across the United States.

- Grid'5000
  Grid'5000 project [87] aims to build a highly reconfigurable, controllable and monitorable experimental Grid platform gathering nine sites geographically distributed in France, featuring a total of 5000 processors.

- D-Grid
  D-Grid project [73] is a German national Grid initiative with the aim of developing a distributed, integrated resource platform for high-performance computing and related services, to enable the processing of large amounts of scientific data and information.

- NorduGrid
  NorduGrid project [97] is a Grid Research and Development collaboration of Sweden, Norway and Denmark, aiming at development, maintenance and support of the free Grid middleware, known as the Advance Resource Connector (ARC).

- CROWN
  CROWN [85], the China Research and Development environment Over Wide-area Network, is a Grid test bed to facilitate scientific activities in different disciplines.

- NAREGI
  NAREGI project [94] is a Japanese national Grid initiative, which aims to research and develop Grid middleware according to global standards to a level that can support practical operation.

[1]Network of Workstations
[2]Distributed Parallel Storage Server
[3]Redundant Arrays of Inexpensive Disks

Table 1.2: Examples of Grid applications

Category	Characteristics	Example
High throughput computing	Hunting for idle resources to increase aggregate throughput	Cryptographic problems [62]
Data intensive computing	Computation with information from many data sources	Digital Sky survey [42] High-energy physics application [66]
Distributed high-performance computing	Large scale applications that need large amount of computing resources	DIS application [69] Chemistry computation [76] Protein Alignment Analysis [176]
On demand computing	Local and Remote resources integration	Medical application [80, 81] Cloud detection [60]
Collaborative computing	Collaboration of multiple partners from multiple disciplines	Collaborative design [25]

1.1.3 Current research challenges facing Grid computing

Though significant progress has been made in Grid computing, various difficulties are expected to cumber the further pavement of Grid computing:

- Resource control and provision
  Normally, resources are shared on Grid infrastructure then employed by various Grid users. Current resource management of Grid computing is based on the traditional multi-programmed machine model, in which resource is controlled and accounted in the granularity of “process” or “job”. Resource management based on small granularity brings complex and excessive problems for job scheduling, migrating and resource accounting.

- Security
  Security is an important topic for Grid computing, while current solutions seem to be too complex and inefficient. In general, Grid users are required identify himself with some proper certificates, work on Grid resources which should pass certain authentications. Authorization is also demanded since users might be recognized after several delegations. It should also be noted that security control is normally based on the granularity of “job”, in other words, security control is activated every time a Grid “job” is submitted to Grid resources.

- Management policy
  Remote execution of Grid jobs on resources sometimes brings issues of management policy. Typical examples might be: Who should provide software licences, Grid users or resource owners? How can Grid users gain some administration privileges to execute their jobs on remote resources?

- Quality of Service (QoS)
  Computational Grid is required to provide QoS for various innovative applications and isolate different user computing environments. QoS guarantee in conventional multi-programmed computing resources is difficult to implement, though some progress [68] has been made.

- Adaptation and customization
  Users are expected to find adaptive and customized computing environments in Grids, which could provide desirable hardware and software configurations for users' requirements. Current Grid computing system however renders deficient support for adaption and customization.

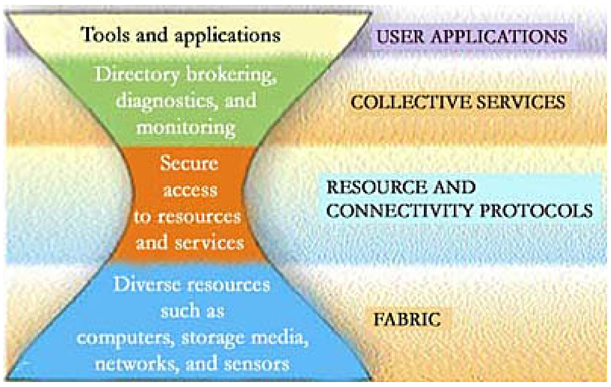

Tools and applications
Directory brokering, diagnostics, and monitoring
Secure access to resources and services
Diverse resources such as computers, storage media, networks, and sensors
USER APPLICATIONS
COLLECTIVE SERVICES
RESOURCE AND CONNECTIVITY PROTOCOLS
FABRIC

Figure 1.1: Hourglass Grid architecture

This Doctoral work identifies key reasons behind the limitations above:

- The basic function of Grid middleware is resource provision. Grid middleware currently, however, is overloaded with too many functions of resource management, data management and security. Despite current Grid middleware being implemented in a distributed manner, this is the bottleneck in Grid computing conception architecture. Figure 1.1 shows Grid middleware offering a set of core services as a basic infrastructure. These core services are then used to construct high-level, domain-specific solutions. Grid middleware core functions implement a number of interfaces to local resource management systems.

- Computing environments of Grid users are directly connected to operating systems of Grid resources with the aid of Grid middleware. When multiple Grid resources communicate with multiple Grid users, especially when this scenario takes place in the granularity of “process” or “job”, multiple control and management functions should therefore be processed.

To solve these limitations, an innovative philosophy for Grid computing should be considered:

- Functions of Grid middleware should be reduced. Grid middleware could keep basic functions such as resource provision, information provision and security control. Other implementations of Grid middleware functions might be as desired by Grid users and implemented at Grid user level.
- Processing control and management in a coarser granularity than “job” or “process” would be a significant improvement.
- Grid user computing environments should be detached from the operating systems of Grid resources.

This thesis therefore presents a new methodology for Grid usage – building multiple virtualized distributed environments for Grid computing, thus overcoming the research challenges facing Grid computing.

1.2 Scope and objective of the research

1.2.1 A brief introduction to virtual machine

Virtual Machine (VM) technology stretches 30 years into the past with IBM’s VM370 [108]. In more recent years there has been a resurgence of interest in utilizing virtual machines for Grid computing as productional virtual machine techniques reach maturity. A virtual machine consists of computer software to reproduce a specific set of computer behaviors and capabilities different from the ones of the computer or operating system in which the software itself is running. Virtual Machine Monitor (VMM) or hypervisor is installed on the host resources to manage virtual machines.

Virtual machines that are served as computing environments can provide the following advantages:

- On-demand creation and customization
  Users can create a customized virtual machine, which provides customized resource allocation for users, e.g., processor, operating system, memory, storage, etc.

- Performance isolation
  Virtual machines can guarantee the performance for users and applications. Applications executed in virtual machines would not find the performance perturbation, which is invoked by simultaneous usage of multiple applications on traditional multi-programmed computers.

- Security
  Employing virtual machines as computing resources can separate user environments from Grid resources, thus prevent malicious attack from users to Grid resources or to other virtual machines. Virtual machine, together with virtual network, build an invisible environment on the Internet, which ensures the security in networking.

Considering various advantages of virtual machines, it is thus a promising solution to contemplate the research issues of Grid computing by adopting virtual machines as computing resources then providing secure, customized, QoS guaranteed virtual environments for Grid users.

1.2.2 Provide a virtual machine or provide a virtual computing environment?

Current research and engineering practices use virtual machines as Grid computing resources and remotely execute Grid jobs on virtual machines.

Normal steps are that:

1. a Grid job arrives at the resource broker of a computing center;
2. the resource broker starts a virtual machine with desired computing environments;
3. the Grid job is executed in the virtual machine;
4. the virtual machine shuts down after the Grid job is finished and returned to the Grid user.

Taking a further step: instead of providing virtual machines directly as the computing resources, a Virtual Distributed Environment (VDE) is provided for users, which contains multiple virtual machines and connected with customized networking. Distributed virtual machines in the VDE could be installed with parallel computing software packages (e.g., MPICH library [96]), distributed operating systems or environments (e.g., CORBA [150] and Cactus [84]), Grid middleware (e.g., Globus Toolkit [135] and UNICORE [101]), and other customized software provision. The virtual machines are linked with customized networking and thus run in a desired network space (e.g., with Virtual Private Network [30] and VNet [120]). As a result coordinated resource sharing and problem solving could be found in the VDE, which is formed with dynamic, multi-institutional, distributed virtual machines. In Grid practice one VDE could be allocated to one Grid Virtual Organization (VO) [42]. The key purpose behind providing multiple VDEs is "**on demand provision of desired computing environments on Grid infrastructures**".

1.2.3 Submit a Grid job or submit a VDE?

It is more interesting to pose the following question:

How should a VDE be operated?

For example, preparing virtual machine disk images of a VDE, to start/shutdown/suspend a VDE, and to store a VDE.

A further scenario could be considered: a computing center or data center provides only thousands of virtual processors. Users could prepare a VDE, which contains numerous virtual machine images. Inside the virtual machine images are installed user computing environments, such as network configurations, security control, software packages and application data. Users can therefore lease virtual processors from the remote computing center, submit their VDE disk images to the remote computing center, execute the VDE, get application results, shutdown the VDE and finally transfer the VDE disk images back again.

The thesis hereby provides a new methodology for using computational Grids – **to submit a VDE to remote Grid resources then execute jobs on the VDE** instead of directly executing Grid jobs on remote resources with the aid of Grid middleware.

1.2.4 Advantages of the new methodology

A lightweight middleware, Grid Virtualization Engine (GVE) is required to support multiple VDEs on Grid infrastructures. GVE is thinner than traditional Grid middleware. It could help solve current limitations facing Grid computing in that:

- Resource management in Grid Virtualization Engine is based on virtual machine level instead of Grid "job" level. Resource management is simpler with coarse granularity of virtual machine. Users can manage resources in the user computing environment inside the VDE.
- Security control is per VDE. The VDE administrator authenticates itself and builds the VDE. VDE Users thus use the VDE with the security policies defined by the VDE administrator and they do not need to contact Grid Virtualization Engine or the underlying Grid resources.
- Virtual machines as computing resources can guarantee QoS and provide customized computing environments for users.

1.3 Main work and contribution of the research

This thesis constructs a lightweight Grid middleware – Grid Virtualization Engine (GVE), to enable multiple VDEs. In order to provide virtual environments for Grid computing the following functionalities of a Grid Virtualization Engine are indispensable:

- Virtualization Service
  Virtualization Service is responsible for managing VDEs for Grid users. It could be deemed as a VMM of Grid version with various enhanced functionalities. In the thesis, a Virtualization Service is developed with standard Web service. It works with VMware ESX server and VMware infrastructure [175].

- Information Service
  Information Service is used to retrieve resource information from distributed host resources and virtual machines. The implementation of Information Service is independent of Grid middleware and adopts industry standards, for example, Web service and Common Information Model (CIM) [129].

With the basic functionalities of Virtualization Service and Information Service, Grid users could build VDEs on Grid infrastructures.

This thesis also presents several typical use cases:

- Grid workflow in a virtual box
  Users can organize virtual machine images, software packages, Grid middleware, Grid workflow system and the workflow application into one VDE, termed a "Virtual Box" and submit it to Grid infrastructures.

  This example demonstrates the simple control on a VDE: resources, middleware, data and applications are packaged together into a Virtual Box and then it could be submitted, executed and migrated easily.

- Virtual e-Science infrastructure
  Users may dynamically deploy an e-Science infrastructure on distributed virtual machines at runtime. This use case gives the justification to dynamic construction and deployment of VDE on-demand.

- Virtual computing center
  Users are able to remotely steer a special type of VDE – virtual computing center, which contains multiple virtual machines and organized by a Grid middleware. Applications could be executed in the virtual computer center. This use scenario illustrates that users can define their own management policies inside a VDE.

This research therefore distinguishes itself from related work in that it:

- investigates new methodologies to harness Grid computing systems by building multiple VDEs on Grid infrastructures, and
- identifies various typical use cases of the VDE on Grid infrastructures.

1.4 Structure of the thesis

The remaining parts of the thesis are organized in the following structure:

- Chapter 2 reviews the background and related work of Grid computing and virtual computing.
- Chapter 3 presents the philosophy of providing multiple VDEs on Grid infrastructures and overviews the Grid Virtualization Engine.
- Chapter 4 and Chapter 5 present two basic functionalities of the Grid Virtualization Engine used to build VDEs for Grid computing: Information Service and Virtualization Service.
- Chapter 6, Chapter 7 and Chapter 8 discuss examples of typical VDE usages, for instance, Grid workflow system in a box, virtual e-Science infrastructure and remote steering of a virtual computing center.
- Chapter 9 presents a scientific Cloud computing effort – the Cumulus project, based on Grid infrastructures and virtualization technologies.
- Chapter 10 concludes the whole research and presents a view of future work.

Figure 1.2 shows the suggested reading order of the thesis.

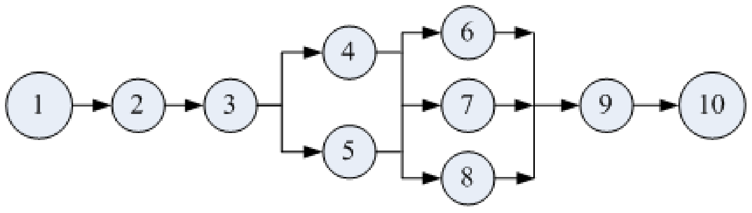


Figure 1.2: Suggested reading order of the thesis

Chapter 2

Related Work

This chapter discusses state of the art of Grid computing and virtual computing environments. Related topics are selected and tailored in the literature review, to build a concrete knowledge background for the Doctoral work.

2.1 Grid computing

2.1.1 Concept of Grid computing

The philosophy of Grid computing can be traced back many years. In 1969, Dr. L. Kleinrock presented his far-sighted idea of Grid computing [57]:

> *"We will probably see the spread of 'computer utilities', which, like present electric and telephone utilities, will service individual homes and offices across the country."*

Dr. Kleinrock's definition might have been prophetic and immature. In 1998, based on the experience of concrete Grid computing projects, I-WAY Project [39] and Globus Project [41], Dr. I. Foster and Dr. C. Kesselman defined the concept of computational Grid as follows [40]:

> *"A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities."*

Grid computing is distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. To attain flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources, a number of challenges are to be expected, such as unique authentication/authorization, resource access, resource discovery. To address these challenges, Dr. Foster and his colleagues proposed the concept of Virtual Organization (VO) and open Grid architecture, in which protocols, services, application programming interfaces and software development kits are categorized according to their roles in enabling resource sharing.

The Virtual Organization is defined as follows [42]:

> *"A set of individuals and/or institutions defined by such sharing rules form what we call a Virtual Organization."*

Based on the concept of Virtual Organization, the concept of computational Grid evolves as follows [42]:

> *"The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations."*

Open Grid Services Architecture

As various Grid computing technologies develop and thousands of Grid computing infrastructures and projects appear, another challenge presents itself. Users need to access Grid systems in a simple and universal manner which demands further technologies for inter-Grid communication.

Standardization is the key to the realization of this Grid vision, so that the diverse components which constitute a modern computing environment can be discovered, accessed, allocated, monitored, accounted for, billed, etc., and generally managed as a single virtual system. This scenario is also interesting and demanding when Grid systems are provided by different vendors and/or operated by different organizations. Global Grid Forum proposes a Service Oriented Architecture, the Open Grid Services Architecture (OGSA) [43], which addresses this requirement for standardization by defining a set of core capabilities and behaviors which are key concerns in Grid systems.

OGSA is the industry blueprint for standard-based Grid computing. OGSA is service oriented, as it delivers functionalities among loosely-coupled interacting services which are compliant with industry-accepted Web service standards. OGSA defines the components, their organizations and interactions, and the overall design philosophy.

In detail, OGSA defines:

- Grid services using a uniform exposed service semantics, and
- standard mechanisms for creating and naming Grid service instances.

OGSA also supports:

- location transparency and multiple protocol bindings for service instances, and
- integration with underlying native platform facilities.

Web Service Resource Framework

In order to provide users with the ability to access and manipulate states of Grid service and promote evolution of Web service, the Web Service Resource Framework (WSRF) [172] is proposed.

A stateful Resource is defined as follows:

- it contains a specific set of state data expressed as XML document,
- it has a well-defined life cycle, and
- it is recognized and processed by one or more Web services.

A *Web Service Resource* (WS-Resource) [28] is defined as follows:

- a Web service Resource is composed of a Web service and a stateful resource;
- a Web service resource is used in the execution of Web service message exchange;
- Web service Resources can be created and destroyed;
- the definition of a Web service resource can be associated with the interface description of a Web service to enable well-formed queries against the sate of a Web service Resource, and the state of the Web service Resource can be queried and modified via Web service message exchanges.

The Web Service Resource Framework is a set of five technical specifications that define the normative description of the WS-Resource approach in terms of specific Web services message exchanges and related XML definitions.

What distinguishes Grid computing

To identify Grid computing from traditional computing paradigms, three check points should be examined as follows [38]:

- Grid coordinates resources that are not subject to centralized control
  There are no central servers which administrate the system. Grid provides coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.

- Grid uses standard, open, general-purpose protocols and interfaces
  Protocols and interfaces, which are used in Grids for authentication, authorization, resource discovery and access, are standard and open and recognized by international organizations, e.g., W3C [173], OGF [153] and OASIS [36].

- Grid delivers non-trivial Qualities of Service
  Grid provides dependable, consistent, pervasive and inexpensive services. In other word, Grid implementations allow resource usage offering various Qualities of Service (QoS), e.g., response time, throughput, security.

Grid system model

To help develop complex Grid systems and software, layered models are built to abstract the architecture of Grid systems. A five-layer hourglass model [42] is proposed to represent Grid architecture (see also Figure 1.1). Components within each layer share common characteristics and can be built on the capabilities and behaviors provided by any lower layers.

The hourglass model contains following layers:

- The *Fabric layer* provides basic Grid protocols that enable Grid applications to share resources;

- The *Connectivity layer* defines core communication and authentication protocols required for Grid-specific network transmission;

- The *Resource layer* builds communication and authentication protocols for the Connectivity layer to define protocols (and APIs, SDKs) for the secured negotiation, initiation, monitoring, control, accounting and payment of sharing operations on individual resources;

- The *Collective layer* contains protocols and services (and APIs, SDKs) that are not associated with any specific resource, but rather interactions across collections of resources;

- The *Application layer* in the Grid architecture comprises the user applications.

Another layered model, the community Grid model, is presented in [61]. It contains following layers:

- *Resource layer* consists of the hardware resources that support the Grid;

- *Common infrastructure layer* consists of the software services and systems, which virtualize the Grid;
- *User oriented middleware layer* contains software packages based on the common infrastructure;
- Users and applications remain at the Grid *Application layer* and access Grid resources and services supported by various lower layers.

2.1.2 Grid middleware

Grid middleware is created by layered interacting packages and sits between Grid infrastructures and Grid applications. It provides Application Programming Interfaces (APIs), services and Software Development Kits (SDKs) for Grid applications. In general Grid middleware provides functions for resource management, information collection and dissemination, data management, and security related issues. In this section, Globus toolkit [135] and UNICORE [101] are investigated as they are utilized in the Doctoral work.

Globus Toolkit

The Globus Toolkit, developed by the Globus Alliance, is an open-source software toolkit used for building Grid systems and applications. A large number of research projects adopt the Globus Toolkit as the Grid middleware for their applications.

The Globus Toolkit includes software packages for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or jointly to develop applications.

The current version of Globus Toolkit is Globus Toolkit 4.0 (GT4). GT4 comprises a set of service implementations (sever side) and associated client libraries (client side) [20]. GT4 provides both Web service (WS) components and non-WS components. GT4 implements a set of predefined services with Web service interfaces:

- GRAM – Grid Resource Allocation Management [19],
- RFT – Reliable File Transfer,
- Delegation,
- MDS (Monitoring & Discovery System):
  - MDS-Index,
  - MDS-Trigger, and

    - MDS-Archive.

- CAS – Community Authorization,
- OGSA-DAI (OGSA Data Access & Integration), and
- GTCP – Grid TeleControl Protocol.

GT4 also provides some pre-WS implementation for GRAM and MDS-Index. Some GT4 services exist without WS interface, e.g., GridFTP and MyProxy.

UNICORE

The development of the UNICORE (**UN**iform **I**nterface to **CO**mputing **RE**sources) system was conceived in 1997 by German Ministry for Education and Research. The aim is to facilitate users, who have large problems in computational science, to utilize various computing resources in different locations [101].

Current UNICORE release, UNICORE 6, is built on the latest Grid service standards, such as official OASIS standard: WSRF 1.2. UNICORE is also compliant with the Open Grid Forum's Open Grid Services Architecture (OGSA).

The UNICORE system contains following components:

- UNICORE client
  UNICORE provides various access methods for users:
    - UNICORE Command Client (UCC) presents users with various command line interfaces such as job submission, file transfer and resource monitoring.
    - UNICORE GUI client helps users submit jobs, monitor resources in a simple graphical environment.
    - Grid Programming Environment (GPE) provides users with Java programming interface to access UNICORE servers and build user defined UNICORE services.
- Web service (WS) frontend
  The UNICORE WS frontend is an reference implementation of OASIS WSRF 1.2 standard. It provides several WSRF 1.2 compliant services, such as:
    - UNICORE atomic services, for instance, *TargetSystemFactory*, *TargetSystem*, *JobManagement*, *StorageManagement*, *FileTransfer*,
    - Registry service,
    - Workflow services, for example, *WorkflowTargetSystem*, *WorkflowJobManagement*, and

  - GridBeanService

- Non Web service (Non-WS) back end
  The non-WS back end contains destination systems and related batch systems and data storage. One computing center could be deemed as a UNICORE site (Usite), which can furthermore include multiple destination systems (*Vsite*s).

A rich core set of functions have been developed in UNICORE that allow users to create and manage complex batch jobs, which could be executed on different systems at different sites:

- Job preparation,
- Job management,
- Data management,
- Resource management,
- Application support,
- Flow control,
- Meta-computing,
- Single sign-on model, and
- Support for legacy applications

2.1.3 Grid workflow system

This section discusses the topic of Grid workflow system, which is the base of Chapter 6 and Chapter 7. A workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal [12, 13]. With support from Grid middleware and Grid infrastructures, the Grid workflow system defines, specifies and manages workflow on the computational Grids [177].

Figure 2.1 shows a general conceptual model of Grid workflow system. Users define and specify workflows in the specification environment, commonly with a Graphical User Interface (GUI). When users model workflows, information from Grid resources and applications running on computational Grids can be retrieved and referred for workflow construction. A Grid workflow engine is responsible for organizing workflows and submitting workflows to Grid middleware via Grid Execution Services.

As a practical discussion of Grid workflow systems, the following important research issues are investigated:

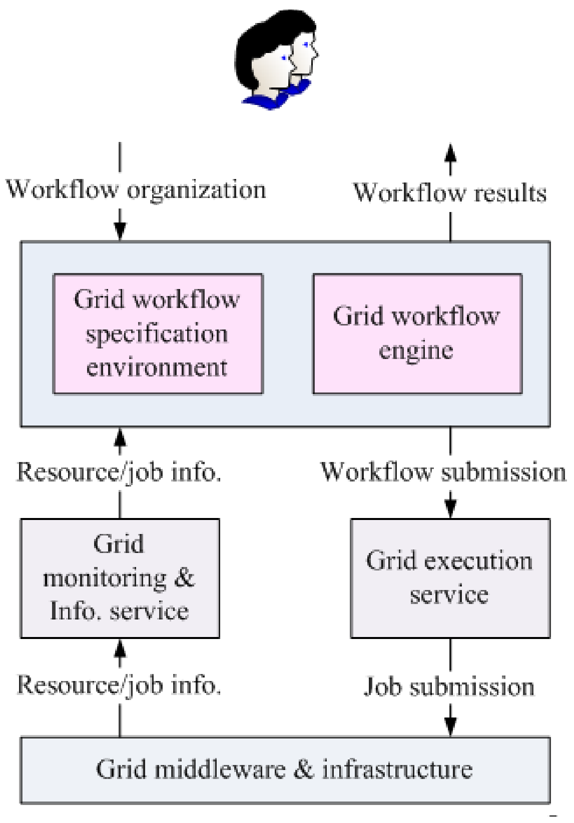


Figure 2.1: General conceptual model of Grid workflow

- Control flow model
  A Grid workflow includes multiple tasks and is executed according to dependencies among tasks. Control flow models the execution order of a workflow. Some Grid workflow systems, for example, UNICORE [101] and GriPhyN [90], adopt Directed Acyclic Graph (DAG) as a control flow model. In DAG based control flow, workflow can be executed in sequential, parallel and conditional patterns. Some other systems, e.g., Triana [122], ASKALON [31], contain rich semantics used for control flow, e.g., the iteration pattern, which allows repeated execution tasks.

- Data flow model
  To run a task of a Grid workflow, input data should be staged to the host where the task is executed and output data are required to be transferred to users or other tasks. To model a data flow, two methods generally exist: the explicit method and the implicit method. With the explicit method, data flow is clearly specified with data source, data destination and transfer protocols. The data transfer process can be modeled as a normal task in the control flow. With the implicit method, tasks only define the required input data and specify the generated output data. Grid workflow engines manage data sets and are responsible for transferring data for tasks. OMII-BPEL workflow [29], which is used

in Chapter 8, utilizes the explicit data flow model.

- Workflow specification
  Two methodologies are used in the workflow specification: the abstract model and the concrete model. In the abstract model, Grid workflow is specified with semantic definition, in which tasks are specified with their functions, abstract interfaces, instead of specific Grid resources and executables. Abstract model offers a simple method for users to define workflows. On the contrary, the concrete model constructs a workflow with detailed execution information, e.g., locations of executables and data transfer protocols. Chimera [22] and Pegasus [23] in GriPhyN Project are example implementations of the abstract model and the concrete model respectively. Various tools and languages are involved in the workflow specification. Fraunhofer Resource Grid [88] uses Petri nets [79] to model the control flow, ASKALON [31] presents rich control flow patterns with UML [168], GridAnt [59] adopts XML as language for workflow representation and Triana [122] allows users to define their own modules and to compose workflow together with existing libraries and software modules. Some special-purpose languages are also developed, for example, GSFL [91], A-WGL [32], and VDL [22]. Almost all recent projects on Grid workflow include a Graphical User Interface for user to construct a workflow. OMII-BPEL project [29] uses BPEL [149] as the specification language.

- Scheduling related issues
  Workflow scheduling decides the starting time and execution hosts for tasks of a workflow. It is well known that scheduling a workflow (or a task graph) in a heterogeneous distributed system is a NP-hard problem [109]. Therefore, scheduling heuristics are adopted to get a near optimal solution.

  For mapping a set of tasks to distributed resources, two taxonomies are used: with/without global workflow information and with/without run-time information:

  - Global scheduling determines the mapping from tasks to resources with the aim of optimizing some function for all tasks of the workflow. Local scheduling, on the contrary, only considers the current task and attempts to find the optimism mapping.
  - Static scheduling constructs a mapping before the execution of workflow on Grid resources. Dynamic scheduling uses dynamic information & static information and plans for workflow at run-time. Artificial intelligence techniques, for example, are investigated in GriPhyN project.

  Various scheduling strategies are employed in workflow scheduling. Most workflow systems adopt performance-aimed strategies, for example, minimization of turn-around time of workflow [59], maximization of the throughput of a workflow system [44]. Some workflow systems, e.g., Nimrod-G [9], considers economic factors and employs market models for workflow scheduling.

- Workflow implementation
  Grid workflow systems contain different models as components for workflow execution.

In GridAnt project [89] and GSFL project [91], for example, the component of workflow execution is Grid service, which is based on different Globus Toolkits version, GT2, GT3 and GT4. For Symphony [67], the component is JavaBean. CAST Project [75] uses parallel CORBA objects as workflow components. Taverna Project [51] and OMII-BPEL project [29] adopt Web services for workflow components. In Triana project [122], users can define their own Java classes as workflow components.

2.1.4 Taxonomy of Grid application

The Doctoral work is evaluated with several Grid applications, such as CMS benchmark (Chapter 6), bio-sequence alignment (Chapter 7), and parallel satellite image processing (Chapter 8).

Grid applications need functionalities of computational Grids to access multiple types of resources shared in computational Grids. On the other hand, Grid computing needs various types of applications to develop, progress, and evolve. This section classifies typical applications for computational Grids.

In the early days of Grid computing, the primary goal for applications of using Grid was to access computing resources and data resources. At this stage, Grid computing is thus resource-centric.

Therefore, Grid applications are initially categorized into the following classes based on how applications employ computing resources/data resources [40]:

- Distributed supercomputing
  Distributed supercomputing applications usually demand extensive computing power, which cannot be fulfilled at a single site. This is the initial reason of I-WAY project [39], which is also the prototype and forerunner of the Globus project. Examples of these applications include distributed interactive simulation [124], which is a technique used for training and planning in military application, and accurate simulation of complex physical processes, e.g., cosmology [145].

- High throughput computing
  In High-throughput computing, computational Grids deal with a large number of independent or loosely coupled jobs, with the goal of maximizing the usage of computing resources. A typical example is the Condor system [64].

- On-demand computing
  On-demand computing applications use computing resources on computational Grids to meet short-term requirements. Requirements of these applications usually cannot conveniently be located on local sites. For example, a computer-enhanced MRI machine and scanning tunneling microscope developed at NCSA use supercomputers for image processing [93].

- Data-intensive computing
  Data-intensive applications focus on synthesizing information from huge data sets, e.g., digital libraries, database, raw data from scientific experiments. Some examples include digital sky survey [6], high-energy physics experiments [178], and remote satellite observation [65].

As the methodology of Grid computing evolves, the concept of Service Oriented Computing has been accepted by Grid communities and collaborative applications become an important type of applications for Grid computing.

Current Grid applications can be categorized based on various primary purposes of applications using computational Grids [4]:

- Community-centric application
  Some applications attempt to create an environment where people or communities work together with certain collaborative objectives. For example, Access Grid [72] is a project which allows interactive video presentation and conferencing from many sites simultaneously.

- Data-centric application
  As discussed above, many applications need to process significant amount of data. Core tasks of these applications are to retrieve, mine, analyze and visualize the data [105].

- Computation-centric application
  Traditional high performance computing applications, common in astrophysics [105], automotive/aerospace industry [77], and climate modeling [86], can benefit from Grid computing considering huge computing resources shared in Grids.

- Interaction-centric application
  Some applications require, or are enhanced by, real-time user interaction. This interaction can take many forms, ranging from decision-making to visualization.

2.1.5 Service Oriented Architecture and e-Science

Service Oriented Architecture (SOA) and e-Science now emerge as hot topics of distributed computing and Grid computing. This section reviews the basic concepts and backgrounds of Service Oriented Architecture and e-Science. The discussion could be found valuable for the work presented in Chapter 4, Chapter 5, and Chapter 7.

Web service and SOA

The World Wide Web Consortium (W3C) [173] defines a Web service [170] as:

"a software system designed to support interoperable machine to machine interaction over a network."

In detail, Web services are application components, which communicate using open protocols, for example, Simple Object Access Protocol (SOAP) standard [163]. Web services are self-contained and self-describing – Web services are described in Web Services Description Language (WSDL) [171] and could be discovered using Universal Description, Discovery and Integration (UDDI) [148].

Web service is gaining popularity because of the benefits it provides. Listed below are some of the key benefits:

- Inter-operability in a heterogeneous environment
  Web service model permits different distributed services to run on a variety of software platforms and architectures, thus enabling inter-operability in a heterogeneous environment.
- Integration with existing systems and legacy applications
  Web services allow enterprize application developers re-use these existing information assets. Legacy codes could be wrapped with Web service to gain popular usage.
- Underlying technology independent
  Web service standards define interfaces and access methodologies of service usage. Web service developers could feel unhindered in selecting various implementation technologies to build their systems.
- Computing and data processing integration
  The client can requires the WSDL definition to effectively exchange data with the service or demand data processing just like pose a normal compute requirement. These benefits allow organizations to integrate applications and data processing with relative ease.

Web services could be deemed as a set of tools that can be used in a number of ways. Common usages of Web service are Remote Procedure Call (RPC) [160] and employing Web service in a Service Oriented Environment.

RPC is a technology that allows a computer program to cause a subroutine or procedure to execute in another memory address space (commonly on another computer in a shared network) without the programmer explicitly coding the details for this remote interaction. In RPC Web services, remote procedures are wrapped with Web service and present as WSDL operations (see also Figure 2.2).

Web services can be used to implement a Service Oriented Architecture (SOA). A major focus of Web services is to make functional building blocks accessible over standard Internet protocols, which are independent of platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network accessible.

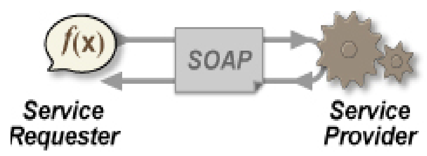


Figure 2.2: RPC of Web Service

Service Oriented Architecture [74] includes the following roles (see also Figure 2.3):

- Service provider
  The service provider creates a Web service and possibly publishes its interface and access information to the service registry, e.g., a UDDI.
- Service broker
  The service broker, also known as service registry, is responsible for making the Web service interface and implementation access information available to any potential service requestor.
- Service requestor
  The service requestor or Web service client locates entries in the broker registry using various search operations and then binds with the service provider in order to invoke one of its Web services.

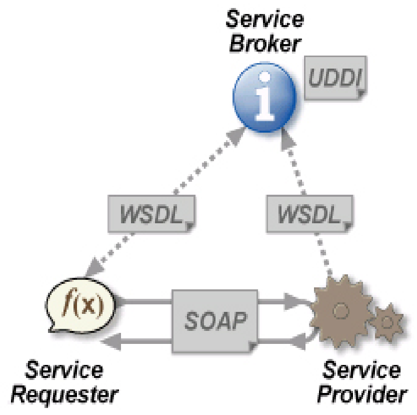


Figure 2.3: Service Oriented Architecture

It is believed that SOA can assist businesses respond more quickly and cost-effectively to the rapidly changing market conditions. SOA promotes the aim of separating users from the service implementations. Services could therefore be executed on various distributed platforms and be accessed across networks. This can also maximize the reuse of services and enable the interoperability of distributed components and entities.

e-Science

e-Science refers to the large scale scientific collaboration, which is normally carried out through globally distributed resources linked by Internet. Typically, a feature of such collaborative scientific enterprizes is that they will require access to very large data collections, high performance computing resources and visualizations back to the individual user scientists. The concept of e-Science was proposed by Dr. J. Taylor in 1999, to describe a large funding initiative starting in November 2000 [27]. Current e-Science projects include social simulations [107], particle physics [178], earth sciences [15, 82] and bio-informatics [121, 176]. Particle physics has a particularly well developed e-Science infrastructure, for instance EGEE project [35], due to the demands for adequate computing facilities for the analysis of results and storage of data originating from the CERN Large Hadron Collider [167].

Considering that "Grid computing focuses on innovative computational applications, enables resource sharing and collaboration across large scale distributed system", it is a natural choice to employ computational Grid as the underlying architecture for e-Science. In our vision, e-Science enjoys the following features:

- e-Science serves innovative applications, which require various types of resources, such as compute resources, data resources, scientific instruments.
- e-Science employs modern distributed computing technologies, for instance Grid and Web service, to build underlying infrastructures. These technologies are normally based on open industry standards and enable scalability, modularity and interoperability.

2.2 Virtual machine and virtual computing environment

2.2.1 Concept of virtualization

Virtualization in a computing context is the process of providing a logical grouping or subset of underlying computing resources. The objective of virtualization is to help access and employ computing resources in ways that bring advantages over the original configuration and organization [113].

In general, virtualization techniques are categorized in the following groups:

- Hardware virtualization
  Hardware virtualization supports a number of discrete computing environments on a single computer, each of which runs an operating system. The computing environment, which appears as a guest operating system for the original hardware, is termed as the Virtual Machine [113].

- Process level virtualization
  Process level virtualization supports the guest process with some runtime software, which could provide a platform independent programming environment. It hides details of the underlying computing infrastructure and allows a program to be executed in the same way on heterogeneous platforms. The guest process is often termed as process level virtual machine [7].

- High level language virtualization
  High level language virtualization could provide a higher level abstraction than the process level. It is implemented using an interpreter. Some popular examples could be found in Java Virtual Machine [63] and Common Language Runtime of .NET Framework [143].

  Another implementation is to offer software library support for existing languages such as C, C++ and Fortran. Typical examples are PVM (Parallel Virtual Machine) [159] and MPI (Message Passing Interface) [144].

- Operating system level virtualization
  A physical server is virtualized at the operating system level to enable multiple isolated and secure virtualized servers on a single physical server. The guest operating system environments are the same operating system as the host system, since the same operating system kernel is used to implement the guest environments.

- Resource virtualization
  Resource virtualization means that multiple resources are aggregated together as a larger and more powerful virtual resource. One example can be checked in the research field of cluster computing. Cluster resource management systems provide users with a single system image of a cluster, i.e., cluster file system, cluster job management and single I/O system image thus making the cluster systems appear as one computing server to users. Some other examples can also be found in the virtualization of data resource: Redundant Array of Inexpensive Drives (RAID) [78] combines many disks into one large logical disk.

- Application level virtualization
  In the application level virtualization, a virtualized environment is built to run user-level programs instead of operation system kernels. The virtual environment is also sometime referred to as virtual private server. It can be thought of as partitioning a single physical resource into multiple virtual servers, which are customized for users with specific software, libraries and other development & execution facilities. Virtual environments are created using the operating system level virtualization approach.

2.2.2 Virtual machine

This section reviews the basic concept of virtual machine. Xen [102] and VMware [175] virtualization technologies are introduced as they are used in the Doctoral work.

A Virtual Machine (VM) is a software artifact that executes other software in the same manner as the machine for which the software is developed and executed [113]. In figure 2.4, the software that supports multiple virtual machines on the same resource is termed as Virtual Machine Monitor (VMM) or hypervisor. The server that provides resources for multiple virtual machines is termed as host resource.

The VMM/hypervisor can provide virtual machines in the following ways:

- Full virtualization
  The virtual machine of full virtualization [108] simulates the complete hardware instruction sets and software stacks from the real systems without modification. The virtual machines provided by the full virtualization technique are completely independent of the nature of the host resource.

- Para-virtualization
  The virtual machine provided by para-virtualization technique [181] does not simulate hardware but instead offers a special set of APIs that requires operating system modifications.

- Native virtualization
  Native virtualization [146], also termed as accelerated virtualization, allows the virtual machine only to partially simulate enough hardware and enables an unmodified operating system to be run in isolation. The guest operating system, however, must be designed for the same type of processor.

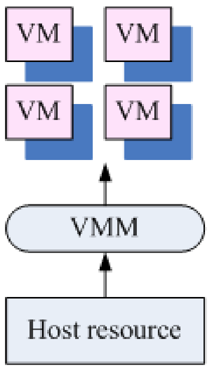


Figure 2.4: Concept of virtual machine

Xen virtualization technology

Xen [8, 102] is a free Virtual Machine hypervisor for various processor architectures. It runs on a host operating system and allows several guest operating systems to be run on the same computer hardware simultaneously. Xen requires to modify the operating system on the host resource. Several modified Unix-like operating systems may be employed as guest systems; on certain hardware, as of Xen version 3.0, unmodified versions of Microsoft Windows and other proprietary operating systems can also be used as guest operating systems.

Xen originated as a research project at the University of Cambridge. It now supports the development of the open source project and also sells enterprize versions of the software. Xen released the first public version in 2003. XenSource, Inc was acquired by Citrix Systems in Oct. 2007. XenSource's products include open source software Xen 3.0, and commercial release such as XenServer Express Edition, XenServer OEM Edition, and XenServer Enterprize Edition.

Xen virtualization provides various distinguished features:

- Para-virtualization
  Para-virtualization is employed by Xen on most processors. It could help Xen achieve high performance compared to various traditional virtualization techniques [104].

- Hardware assisted virtualization which allows for unmodified guests
  With the help of Intel Corp. and AMD Inc., Xen 3.0 manages a common abstraction layer on various processor architecture and enable unmodified guest operating systems to run within Xen virtual machines. It allows proprietary operating systems (such as Microsoft Windows) to be virtualized since the guest system's kernel does not require modification.

- Virtual machine migration
  Xen virtual machines could be "live migrated" between host resources in a LAN environment without loss of availability. During this procedure, virtual machine memories are iteratively copied to the destination host resource without stopping its execution. It is only demanded to pause the virtual machine in a short period, e.g. 60 - 300 ms, to perform final synchronization. Then the virtual machine could begin executing at its the destination host resource. The "live migration" thus provides an illusion of seamless migration.

VMware virtualization technology

VMware [175] is a well-known developer of advanced virtualization software products for x86-compatible computers. VMware products include:

- Desktop virtualization: VMware Workstation,

- Server virtualization:
  - VMware Server (formerly called “GSX Server”),
  - VMware ESX Server, and
  - VMware Infrastructure, which virtualizes and aggregates industry-standard servers and their attached network & storage into unified resource pools.

VMware software provides a completely virtualized set of hardware to the guest operating system. VMware software virtualizes the hardware for a video adapter, a network adapter and hard disk adapters. The host provides pass-through drivers for guest USB, serial and parallel devices. In this way, VMware virtual machines become highly portable between computers, because every host appears almost identical to the guest. In practice, a systems administrator can pause operations on a virtual machine guest operating system, move or copy that guest operating system to another physical computer, and there resume execution exactly at the point of suspension.

2.2.3 Virtualization technology in distributed computing

This section reviews the pioneering projects that use virtualization technology in distributed computing. Although these projects differ from our work in various concerns such as objective, middleware and virtual machine technology, they still remain good references for the Doctoral thesis.

Globus Virtual Workspace Service

A Virtual Workspace [56, 55] is an abstraction of a computing environment which are dynamically available to authorized clients via invoking Grid services. The abstraction assigns resource quota to the execution environment on deployment (for example processor or memory) as well as software configuration aspects of the environment (for instance operating system installation or provided services). The Workspace Service allows a Globus client to dynamically deploy and manage a computing environment.

The Globus Virtual Workspace Service (GVWS) provides the following access interfaces [56, 55]:

- The *Workspace Factory Service* has one operation called *create*, which has two required parameters: workspace metadata and a deployment request for that metadata.
- After it is created, the workspace is represented as a WSRF *resource*. The workspace could be inspected and managed through the *Workspace Service* operations .
- *Group Service* allows an authorized client to manage a set of workspaces at one time.

- *Status Service* offers the interface through which a client can query the usage data that the service has collected about it.

Violin: Virtual Inter-networking on Overlay Infrastructure

Violin [54, 53] is a virtual network technology, which creates high-order virtual IP networks and offers some interesting advantages:

- on-demand creation of both virtual machines and the virtual IP network which connects virtual machines,
- customization of the virtual network topology & services, operating system services, and application services, packages, and libraries,
- achieving binary compatibility by creating the identified runtime and network environment, under which an application was originally developed, and
- containment of negative impact by isolating virtual network address space.

The In-VIGO system

The In-VIGO system [1] contains three layers of virtualization in addition to the traditional Grid computing model:

- The first virtualization layer consists of virtual resource pools. Primitive components of a virtual computing Grid are located in this layer, for example, virtual machines, virtual data, virtual applications and virtual networks. In contrast to the traditional process of allocating applications to resources, this layer manages jobs across administrative domains, physical machines and local software configurations. This means that tasks are mapped to virtual resources and only virtual resources are managed across domains and computing infrastructures.
- Grid applications in the second layer are wrapped as Grid services, which can be connected as needed to create virtual Grids. Therefore, managing the execution of the underlying Grid applications is decoupled from the process of utilizing and composing services. Various Grid computing methodologies can be adopted to execute Grid applications. The implementation details of these mechanisms are hidden when the Grid enabled applications are encapsulated and composed as Grid services.
- In order to enable displaying by different access devices, the third layer aggregates Grid services by exporting their virtualized interfaces. This layer, therefore decouples the process of generating interfaces of services from the process of delivering them on specific devices.

Virtuoso: a system for virtual machine marketplaces

The Virtuoso system [110] is developed by Northwestern University and aims to create a marketplace for resource usage. Resource providers could sell their resources to consumers in the form of virtual machine and virtual network.

In the Virtuoso system, the user acquires a remote virtual machine, configured with desired processor type, memory size, and data storage resources. The resource consumer could install and configure whatever software demanded on the virtual machine, for instance, operating system, compiler or software libraries. A virtual network, VNET [120], links the resource consumers virtual machines together efficiently and makes them appear, regardless of wherever they currently are, to be directly connected to the resource consumer's local network environment. VNET is an adaptive virtual network that can use inter virtual machine traffic pattern inference, virtual machine migration, overlay topology and routing rule manipulation, and resource reservations to optimize the performance of a distributed or parallel application running in the resource consumer's virtual machines.

COD: Cluster-on-Demand

The Cluster-on-Demand (COD) [11, 52] project implements a virtual cluster and with the objective to separate the cluster usage from the cluster management. The virtual cluster consists a subset of cluster nodes configured for some common purpose, associated with user accounts and storage resources, user specified software environment and a private network space. When a configuration is defined, with automatic application to a cluster node, thus makes it easy to redeploy cluster nodes among user communities and software environments under programmatic control. Two levels of administration are introduced in the virtual cluster. It is declared that the virtual cluster offers an opportunity to employ cluster resources more efficiently while providing better services to users.

2.3 Summary

This chapter investigates the related work of Grid computing and virtual computing environment, which builds a solid background to the Doctoral thesis. On the part of Grid computing, topics such as Grid computing concept, Grid middleware, Grid workflow system, Grid application, SOA and e-Science are discussed. Concerning virtual computing environment, this chapter surveys various research fields, for example, virtualization concept, virtual machine, and virtualization technologies for distributed computing.

Chapter 3

On Demand Virtual Environment Provisioning: an Innovative Philosophy for Grid Computing

3.1 Research issues of Grid computing

Current Grid users employ Grid resources by submitting jobs to remote computers. Grid middleware, operating systems and software packages & libraries on remote resources constitute the computing environments for Grid users. It therefore could be concluded that users' computing environments are directly plugged into operating systems of the Grid resources.

Great burdens are afforded to Grid resource providers and Grid middleware:

- Every time a job is submitted, user authentication and authorization has to be carried out. To reach single sign-on, Grid middleware has to provide complex policies and schemes for security control and user delegation.
- Resource management becomes a bottleneck for the Grid middleware. Huge efforts have been carried to manage various types of resources to fulfill different kinds of application domains.

Much effort on security control and resource management, which is application specific, has to be mapped to heterogeneous Grid resources. For example, a workflow application [10] requires different resource management functions from a parameter sweep application [119]. A community-centric Grid application [72] certainly has a security control scheme, distinct from a data-centric application [105]. These functions, which should be decided and deployed in the user level, are currently implemented inside Grid middleware functionalities.

Grid users could expect various disadvantages with the forementioned methodology:

- Users expect customized computing environments with special software and hardware configuration or the abilities to configure such environments. However, multiple Grid users share the same resources. It is therefore hard to balance different users' requirements. Even when a Grid user monopolizes a resource, normally it is not possible to also offer the user with administrative privileges to configure the resource.

- Grid users normally require performance guarantees for executing their applications, e.g., CPU bandwidth, memory allocation. In the traditional multi-programmed computer model, multiple users share the same resource with local users of the Grid site. Grid users have to suffer from the performance fluctuation when executing jobs on Grid resources. Performance guarantee is difficult to attain in the current Grid computing model.

We identify the reasons for these limitations:

- Grid middleware is loaded with too many functionalities, most of which should be moved to and implemented in the Grid application level. It is argued that Grid middleware is only responsible for providing basic functionalities of resource provision, information provision and security control. For example, current support of Grid workflow, MPI [144] applications should be moved to user level environments.

- User computing environments are directly interposed into the operating systems of Grid resources, thus producing a number of interfaces between Grid resources and users environments, for example, process invocation, security control and information exchange. As the user computing environment is attached to the operating system of a Grid resource, customization and performance isolation are difficult and demand additional effort.

- Users harness Grid resources with a fine-grained "job" or "process". Therefore, every operation on this fine granularity would invoke a set of functions of the Grid middleware. Furthermore, it is difficult to provide customized environments and guarantee performance with the fine granularity in a traditional multi-programmed computing system.

3.2 Virtual machine as Grid computing resource

In recent years the Grid community has shown research interests to employ virtual machines for Grid computing. In general, Grid users can benefit from virtual machines in following aspects [34]:

- Security
  Virtual machine technology makes security control easier by classifying as two level: system level security control and user level security control. At the system level, host resource protects the virtual machine image as a complement identity. Virtual machine administrator manages the access control inside the virtual machine at the user level.

- Performance isolation
  Virtual machines can guarantee the performance for users and applications. Applications executed in virtual machines would not find the performance perturbation, which is invoked by resource competition of concurrent processes on traditional multi-programmed computers.

- On-demand creation and customization
  Virtual machine administrators can create and customize virtual machines, which therefore could render desired resource provision for users, e.g., operating system, memory, storage, etc. Virtual machines may help build reconfigurable IT infrastructures. This is an attractive advantage for software development or test.

- Legacy system support
  Virtual machines could support compatibility of legacy applications without recompilation or relinking. Virtual machine can also support entire legacy environments, such as hardware, operating system, and applications, by on-demand creation of desired virtual machines.

- Administration privileges and site autonomy
  Users of virtual machines could obtain the "root" privilege in case that each user of the hosting resources is allocated with a virtual machine. This scenario alleviates the task of system administrator and gives flexibility for application users.

  As virtual machine can be started, shutdown and migrated dynamically, it is therefore easier to manage virtual machines in a computing center than a large amount of jobs. The computing center that provides virtual machine also can keep the site autonomy. For example, computing center is free from supporting software licences to users.

- Resource control
  In traditional multiprogrammed environment, resources are controlled per "process". One user or one application could contain multiple processes. Therefore resource control in granularity of "process" is inappropriate. One virtual machine could be allocated to one user or one application. It is thus effortless to account and control the resource usage.

Therefore employing virtual machines as computing resources could be a promising solution to overcome the various research challenges faced by Grid computing.

3.3 Build multiple VDEs on shared Grid infrastructures

3.3.1 Concept of Virtual Distributed Environment

Normally we could expect to envisage the following methodologies of employing virtual machines as computing resources for computational Grids:

- Direct usage
  Computer servers and clusters as host resources are installed with virtual machine hypervisors. When a Grid job arrives, host resources prepare a virtual machine to execute the Grid job (see also Figure 3.1). This scenario occurs in normal Grid compute sites.

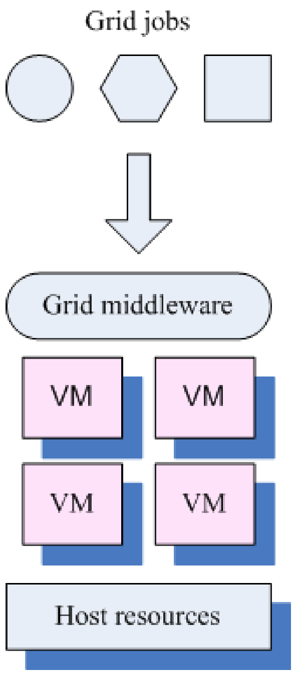


Figure 3.1: Direct usage of virtual machines for Grid computing

- Structured usage
  Computer sites organize virtual machines for incoming jobs. The incoming Grid jobs are structured as, for instance, gang jobs [3] or a workflow [177]. The virtual machines could be organized, for example as a virtual computer cluster or as a Grid gateway, file server, Grid portal, where various Grid functionalities could be implemented. This paradigm could be deemed as an advanced methodology of using virtual machines for Grid computing (see also Figure 3.2).

This chapter defines the term Virtual Distributed Environment (VDE) (see also Figure 3.3) as follows:

- a VDE contains multiple virtual machines, which could be linked by normal networking or virtual networking;

- a VDE is installed with some network protocols or middleware, for distributed computing, which manage the distributed virtual machines in the user level;

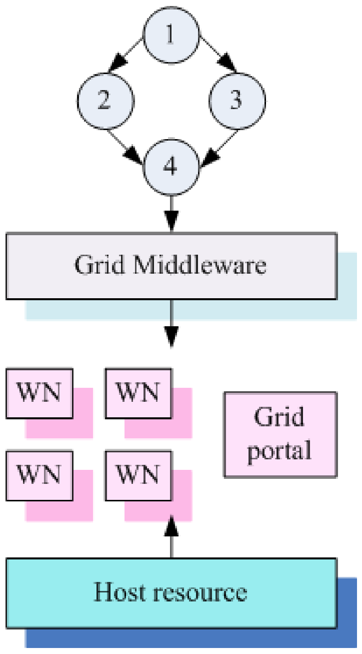


Figure 3.2: Structured usage of virtual machines for Grid computing

- the VDE administrator, who creates the VDE, is responsible for managing the VDE, such as job scheduling, user authentication, and data movement inside the VDE.

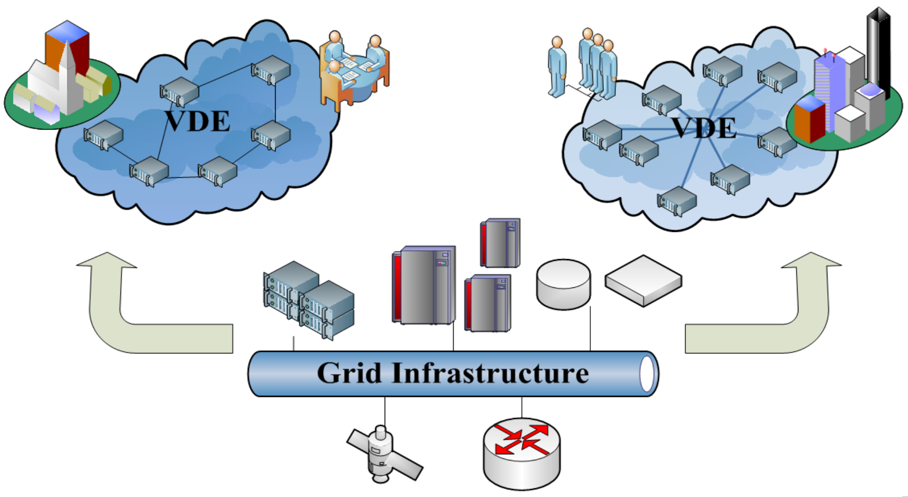


Figure 3.3: Provide multiple VDEs on shared Grid infrastructures

This section therefore proposes a new philosophy for Grid usage based on the VDE concept:

- Virtual machines are used as computing resources for Grid applications. Users can build and operate on virtual machines on-demand, and thereafter a VDE, which contains multiple virtual machines.

- Users submit jobs to VMs or VDEs, which are provided by Grid infrastructures. Users can, furthermore, submit pre-configured VMs or VDEs to remote Grid resources for execution.
- The VMs and VDEs on Grid infrastructures should be supported and managed by a lightweight middleware, which only offers basic functionalities such as resource supply, information provision and security control. The Doctoral thesis implements a prototype of lightweight middleware – Grid Virtualization Engine (GVE).

3.3.2 Virtualization philosophy of VDE

Providing multiple VDEs on a shared Grid infrastructure could be deemed as an extension of offering multiple VMs on a host resource. In Table 3.1 some comparisons between providing VMs and providing VDEs are listed.

Table 3.1: Comparison of VM provision and VDE provision

Item	**VM provision**	**VDE provision**
Computing environment	VM	VDE
Enable software	VMM or hypervisor	Grid Virtualization Engine
Support platform	Host resource	Grid infrastructure
Customization	Operating system, software packages & libraries, user environments	Networking, operating systems, software packages & libraries, security control, distributed environments

The new methodology of Grid usage can help solve the issues faced by Grid communities:

- Resource management is now based on the coarse granularity – “virtual machine” instead of the fine granularity – “Grid job”. Resource management becomes simpler with coarser granularity. Users could develop application specific or user community specific resource management solutions inside their own VDEs.
- Security control is based per VDE. The VDE administrator authenticates itself and builds its VDE. Users thus utilize the VDE with the security policies defined by the VDE, and they do not need to contact with Grid Virtualization Engine and Grid resources.

- Virtual machines as computing resources can offer performance guarantee and provide customized computing environments for users.

Common underlying ideas of virtualization fall into two categories:

- Resource multiplexing
  A middleware is posed between resources and users and it produces an illusion for each user that he monopolizes the resource without awareness of the existence of other users. Normally, a translator is needed to map user level logics to resource level stuffs. Examples could be found in Virtual Private Network (VPN) [70], Virtual Machine, and virtual memory system in modern operating systems.

  It could be summarized as: **multiplex one resource to multiple resource images: 1 resource → M resource images**.

- Resource consolidation
  Resource consolidation is generally used for managing discrete resources to hide low level technical details and provide an easily used, uniformed resource interface to users. Various instances could be enumerated in modern parallel/distributed computing systems, e.g. a cluster operating system.

  It could be summarized as: **consolidate multiple resources to one resource image: M resources → 1 resource image**.

The methodology of building multiple VDEs on a shared Grid infrastructure paves a further step and combines the ideas of **resource multiplexing** and **resource consolidation** together. Access distributed Grid resources via Grid middleware could be deemed as **resource consolidation**. To build multiple VDEs on shared Grid infrastructure via the GVE could be considered as **resource multiplexing**. Therefore a novel usage methodology of Grid computing is proposed: **resource consolidation + resource multiplexing**: **M resources → M resource images**.

3.3.3 Sample usages of the VDE

This section investigates some typical use scenarios of the VDEs on shared Grid infrastructures:

- Grid workflow in a Virtual Box
  A Grid workflow system is a VDE that contains:
  - compute resources which are virtual machines,
  - Grid middleware that organizes distributed virtual machines together and provide functions, for example, workflow execution.

  - workflow applications that contains executables, data sets and other workflow related files, and

  Users can place virtual machine images, Grid middleware and applications together into a virtual box, which is a bundle of discrete data files. Then Grid users are able to remotely submit, execute and control the Grid workflow in a Virtual Box.

- Dynamically build virtual e-Science infrastructure
  An e-Science infrastructure [27] could be, for example, an OMII-BPEL workflow engine [151] and GridSAM [118] as Grid middleware. Users are capable of dynamically organizing a virtual e-Science infrastructure:

  - request and customize distributed virtual machines with required software, such as GridSAM Web service,
  - organize workflow with ActiveBPEL engine and execute the workflow on distributed virtual machines via GridSAM.

- Virtual computing center
  Another interesting scenario is that remote computing center provides a great deal of virtual processors, which form a virtual computing center. Users thus have the ability of building a middleware (e.g., UNICORE) on the virtual computing center, then steering the virtual computing center. This mode brings a huge amount facilities to harness remote computing resources.

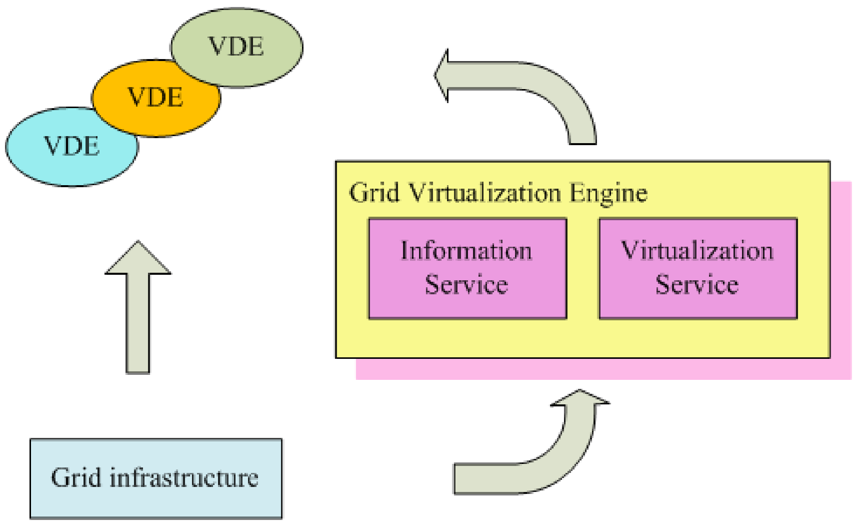


Figure 3.4: Grid Virtualization Engine

3.4 Grid Virtualization Engine

3.4.1 Overview of Grid Virtualization Engine

To build multiple VDEs on Grid infrastructures, a lightweight middleware, Grid Virtualization Engine (GVE), is designed and implemented in the Doctoral work.

The GVE aims to provide users with following functions:

- users can remotely create, operate, configure and manage virtual machines;
- users can create, customize and manage a VDE with multiple virtual machines on wide area networking.

The GVE offers two services:

- An Information Service collects virtual machine information and transfers them to high level Grid services.

  Our implementation enjoys following features:

  - Lightweight implementation
    The Information Service, which runs in virtual machines and host resources, is implemented with lightweight Java agent and employs VMware CIM protocols [129].
  - Grid middleware independent
    The Information Service does not reply on any specific Grid middleware. It can provide virtual machine information to multiple Grid high level services.
  - Translation between different information schemas
    Virtual machine information, in general, follows CIM information schema. High level Grid service, on the other hand, normally uses GLUE schema. Our work can translate CIM based virtual machine information [129] to Grid high level services, which use GLUE schema [136].
- A Virtualization Service provides functions of manipulation on virtual machines from computational Grids, for instance:
  - Operations on virtual machines such as creation, destruction, startup, shutdown, etc.
  - Automatic computing environment configuration, such as software installation, network configuration.
  - Build multiple VDEs with different configurations for Grid computing.

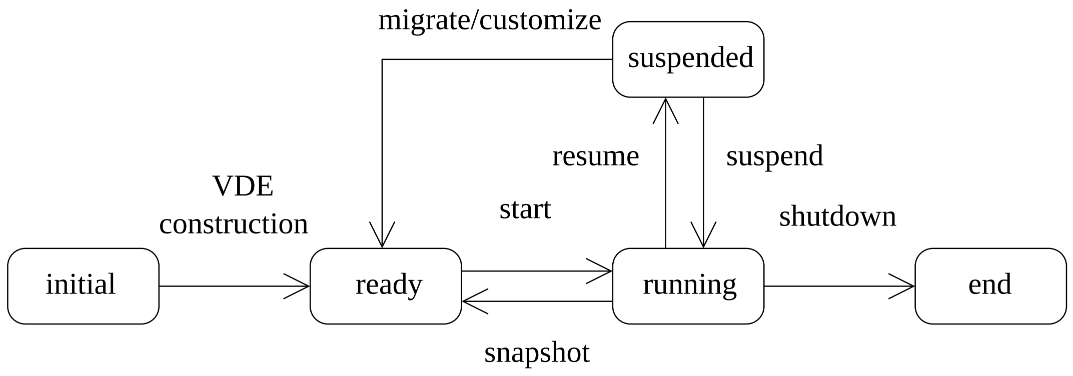


Figure 3.5: VDE states

3.4.2 VDE life cycles

With these functions, GVE offers various VDE operations to change VDE states (see Figure 3.5). The VDE operations are implemented and demonstrated in the VDE sample usages.

VDE operations include:

- VDE is changed from the starting state of "initial" to the "ready" state by VDE construction operation:
  - build multiple virtual machines for a VDE;
  - install and configure a distributed resource management system on various virtual machines, for example, a Grid middleware such as UNICORE, a cluster operating system, or distributed communication interface, e.g., virtual private network;
  - customize the VDE with other considerations, for example, user management and policy control.
- The "ready" VDE could be converted to the state "running" by VDE start.
- After VDE is finished execution, the "shutdown" operation is executed and VDE arrives the "terminated" state.
- The"snapshot" operation could store current working status of the "running" VDE, which is then marked as the "ready" state for execution.
- The "running" VDE could be suspended and the VDE takes the "suspended" state.
- The "suspended" VDE could be restarted to "running" by VDE "resume" operation.
- The "suspended" VDE could be customized or migrated to a new location, then becomes "ready" for future execution.

3.4.3 Target system model

This section defines the Grid system model used for the thesis, which contains distributed Compute Sites interconnected by networking.

Each Compute Site logically consists of the following levels:

- The Compute Site provides an *access service* which allows remote users to access resources of the computing center. The access service could be offered by existing Grid middleware, a portal, Web services, or any functionalities that support remote steering. Information Service and Virtualization Service are developed and integrated in this level.

- At the middle level exist *virtual machines* that are backed by host resources. These virtual machines form VDEs. Information Service and Virtualization Service operate virtual machines in this level.

- Fabric level contains various *host resources* or servers, which are installed with virtual machine hypervisors. Host resources offer multiple virtual machines. Back ends of Information Service and Virtualization Service are implemented in this level with support from VMM APIs and SDKs.

3.4.4 Test bed

The Doctoral work is carried on the SCC (Steinbuch Centre for Computing) test bed which is shown in Figure 3.6, Table 3.2 and Table 3.3.

The test bed contains two parts:

- Computer servers in IWR/FZK
  Computer servers in IWR/FZK (Institute for Scientific Computing, Research Center Karlsruhe) are installed with various VMMs, such as Xen server, VMware server and VMware ESX server.

- A compute cluster in RZ/UKA
  The compute cluster in RZ/UKA (Computing Center, University Karlsruhe) consists 1 head node and 12 worker nodes, all of which are installed with Xen server. The head node backs 8 virtual machines, which offer various cluster/Grid functionalities, e.g., file server, monitor server and Grid portal. Each worker node provides one virtual machine, which is used to execute Grid jobs. There is a batch scheduler in the cluster to schedule local jobs on worker nodes.

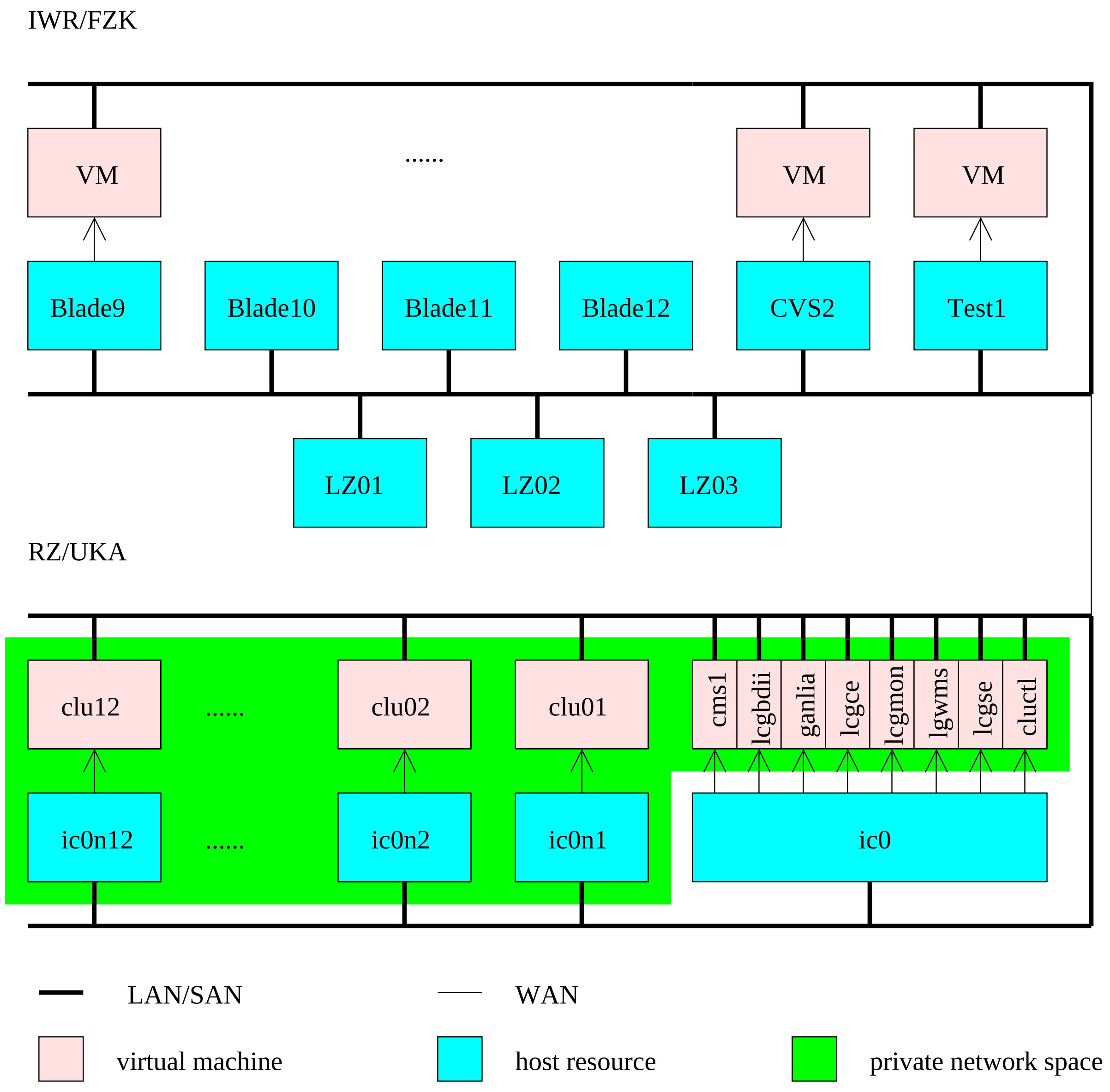


Figure 3.6: Test bed

3.5 Summary

In this chapter, current research issues and challenges faced by Grid computing are identified and discussed. This chapter, therefore, presents the philosophy of building multiple VDEs on shared Grid infrastructures. The new philosophy declares that it could solve the problems and challenges that Grid community is envisaging. Three typical scenarios of using VDEs are provided, which will be investigated later in Chapter 6, Chapter 7 and Chapter 8. An overview of virtualization middleware – Grid Virtualization Engine (GVE) is presented in this chapter to

Table 3.2: IWR/FZK test bed description

Resource name	**Resource description**
Blade9	VMware ESX server
	IBM eServer Bladecenter HS20
	2 × Intel® Xeon™ CPU 3.00 GHz
	2.68 GB RAM
	Linux version 2.4.21-37.0.2.ELvmnix (Red Hat Linux 3.2.3-34)
Blade10	VMware ESX server
	IBM eServer Bladecenter HS20
	2 × Intel® Xeon™ CPU 3.00 GHz
	5.08 GB RAM
	Linux version 2.6.16.46-0.12-smp (SUSE Linux)
Blade11	Xen server
	IBM eServer Bladecenter LS120
Blade12	2 × AMD Opteron™ Processor 250
CVS2	VMware Infrastructure 3
Test1	Xen Enterprise Server
	2 × Intel® Xeon™ CPU 2.4 GHz
	3.0 GB RAM
	Linux version 2.6.18-8.1.8.e15.xs4.0.125.163xen (Red Hat Linux 4.1.1-52)
LZ01	Linux Server
	1 × Intel® Celeron™ CPU 2.00 GHz
	256 MB RAM,
	Linux version 2.6.13-15-default (SUSE Linux)
LZ02	Linux Server
	1 × Intel® Pentium™ M processor 1.5 GHz
	512 MB RAM
LZ03	Linux Server
	1 × AMD Athlon™ 64 Processor 3500+
	2 GB RAM
	Linux version 2.6.18.2-34-default (SUSE Linux)
VM	Virtual machine
	Worker node

enable VDEs on shared Grid infrastructures.

Table 3.3: Test cluster at RZ/UKA

Resource name	Resource description
Test cluster	Compute cluster Maui batch scheduler ic0 = head node ic0n01∼ic0n12 = worker node
clu01∼clu012	Virtual machines Worker nodes
cluctl	Virtual machine Debian 4.1
lcgse	Virtual machine Scientific Linux CERN Release 3.0.8 Storage Element
lcgwms	Virtual machine Scientific Linux CERN Release 3.0.8 Resource Broker
lcgmon	Virtual machine Scientific Linux CERN Release 3.0.8 Grid Monitoring Server
lcgce	Virtual machine Scientific Linux CERN Release 3.0.8 Computing Element
ganlia	Virtual machine Debian 4.1 Ganglia Cluster Monitoring Server
lcgbdii	Virtual machine Scientific Linux CERN Release 3.0.8 BDII Server
cms1	Virtual machine Scientific Linux CERN SLC release 4.5 Portal

Chapter 4

A New Information Service for Monitoring Virtual Environments

4.1 Overview

In general, an Information Service for a distributed system should provide the following functions [48]:

- efficient delivery of state information from any sources,
- respond robustly to the failure of a component,
- be free of time stamp failures,
- support a few mandatory discovery and enquiry mechanisms,
- support a rich set of discovery and monitoring strategies, such as hierarchical resource groupings, multiple naming schemes, and search strategies, and
- support robust authentication and policy framework.

In traditional Grid computing system, Compute Site contains an Information Service that retrieves resource information from Computing Element and Storage Element directly.

When virtual machines are employed for VDEs, some specific requirements are posed on the Information Service:

- Information Service should be scalable and robust regarding dynamic startup/shutdown of virtual machines;
- The information provider, which runs inside the virtual machine and collects resource information, should be lightweight, portable and manageable.

- A portable interface should be constructed between the Information Service and the information provider of virtual machines.

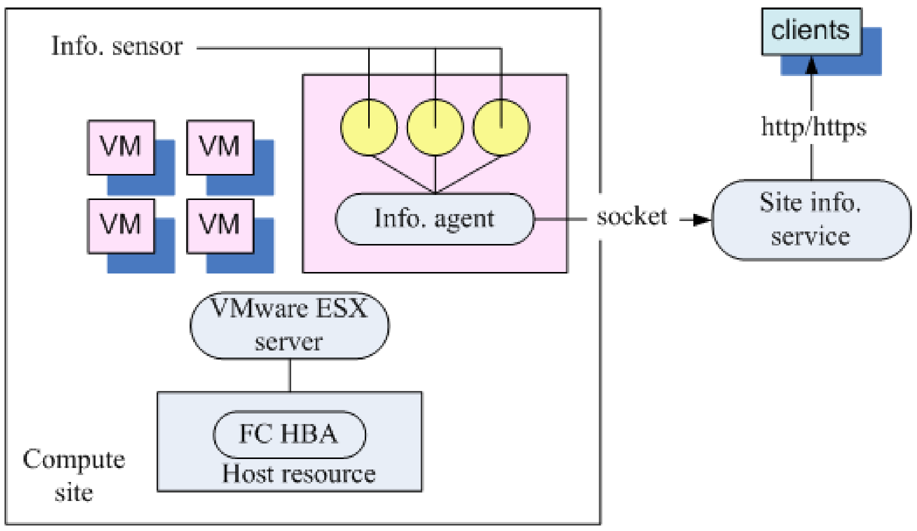


Figure 4.1: Overview of Information Service

This chapter discusses the design and implementation of the Information Service, which provides resource information for distributed virtual machines on computational Grids. The Information Service is designed in a hierarchical structure (see also Figure 4.1):

- A lightweight Java Information Agent is developed and collects resources information as an Information Collector for virtual machines;
- WSRF-compliant Site Information Service is developed and retrieve resource information from Information Agents.

4.2 Information Agent

4.2.1 Introduction

The Information Agent (IA) is an autonomous system, which is developed in Java classes and runs as a daemon in the virtual machine.

Clients can:

- query Information Agent locally from command line,

- obtain the resource information from Information Agent via SOCKET message communication, or
- subscribe to Information Agent and receive updated information periodically.

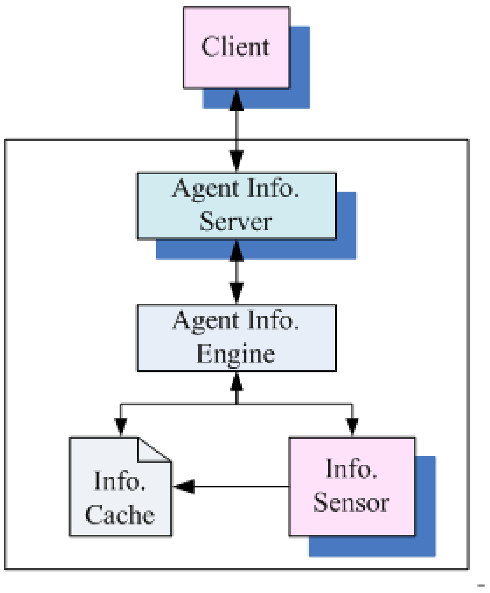


Figure 4.2: Information Agent

Information Agent contains the following components (see also Figure 4.2):

- Agent Information Server
  The Agent Information Server receives inputs from clients, executes queries to the Agent Information Engine, and returns resource information to clients. In implementation, Java main class of Information Agent creates a thread as an Agent Information Server to process the query when a new query arrives.
- Agent Information Engine
  The Agent Information Engine is responsible for processing the query transferred from the Agent Information Server. The Agent Information Engine firstly queries the Agent Information Item Cache in the memory. Each Agent Information Item has a time stamp and timeout value. The Agent Information item values expire as time elapses more than the timeout value. When the information in the cache is missed or expired, the Agent Information Engine then runs Agent Information Sensors to get the information.
- Agent Information Sensor
  The Agent Information Sensor is devoted to retrieve some specific resource information. For example, memory sensor provides memory information such as free memory size,

total memory size and swap memory size. In the implementation, the Information Agents are *perl* scripts to obtain memory, CPU, operating system information. It is also possible to acquire the application runtime information with user-defined Information Sensors.

- Agent Information Cache
  Each Agent Information Sensor can provide several Agent Information Items. For instance, CPU sensor delivers information items such as CPU type, CPU number and CPU load. The Agent Information Cache is a block of memory allocated to store values of Agent Information Items. Each time the information sensors are executed, the information item cache is updated.

Table 4.1: Mapping table for Information Sensor/Item

Agent Information Item	**Agent Information Sensor**	**Sensor source**	**timeout**(s)
cpu.count	cpuinfo	/proc/cpuinfo	-1
cpu0.cache			
cpu0.model			
cpu0.freq			
cpu.load1	cpuloadinfo	/proc/loadavg	10
cpu.load5			50
cpu.load15			150
host.name	systeminfo	/bin/uname	-1
os.name			
os.version			
mem.total	meminfo	/proc/meminfo	-1
mem.free			10
mem.buffers			10
mem.cached			10
swap.total			10
swap.cached			10
swap.free			10

4.2.2 Implementation

The Agent Information Server is constructed when the Information Agent is started. The Agent Information Server reads two configuration files:

- Information Agent configuration file
  Configurations for the Information Agent exist in the configuration file, e.g., the port number via which it listens for query.

- The Agent Information Sensor and the Agent Information Item configuration file
  The Agent Information Server reads the configuration files and builds a table in memory, which maps Agent Information Items to Agent Information Sensors. An Agent Information Sensor configuration file maps the Agent Information Sensor name and the executable. An Agent Information Item configuration file maps the Agent Information Sensor and the Agent Information Items provided. A typical mapping table is shown in Table 4.1. Based on the table, the Agent Information Server can map the query to the Agent Information Item – Agent Information Sensor pair and related information, e.g., information source and time out for the item.

On receiving the query request, the Information Agent starts up a thread to precess the query, which is the Agent Information Server. It checks the syntax of the query and locates the related Agent Information Item and Agent Information Sensor, which provide resource information for the query.

Table 4.2: Agent Information Item Cache

Agent Information Item	**Value**	**Time stamp**
cpu.count	1	-1
cpu.cache	512	-1
cpu.model	AMD Athlon$^{(tm)}$ 64 Processor 3500+	-1
cpu.freq	2200.106	-1
cpu.load1	0.26	1160644745341
cpu.load5	0.20	1160644865341
cpu.load15	0.12	1160645165341
host.name	IWR-LIZHE3.ka.fzk.de	-1
os.name	Linux	-1
os.version	2.6.16.21-0.15	-1
mem.total	1927168	-1
mem.free	779360	1160644725329
mem.buffers	66736	1160644725329
mem.cached	592624	1160644725329
swap.total	530104	1160644725330
swap.cached	0	1160644725329
swap.free	779360	1160644725330

The Information Agent starts the Agent Information Engine to retrieve the value. In the Agent Information Engine, an Agent Information Item Cache is maintained in memory. A typical Information Item cache is organized as Table 4.2. If the Agent Information Engine cannot locate the Agent Information Item in the cache or the value in the cache is expired, the Agent Information Engine executes the corresponding Agent Information Sensors, which have been identified by the Agent Information Server, and retrieves the resource information.

After the execution, the Agent Information Sensor also updates other related information items in the Agent Information Cache. After acquisition of resource information from the Agent Information Cache or Agent Information Sensors, the Agent Information Engine transfers the values to the Agent Information Server, which finally returns resource information to the client.

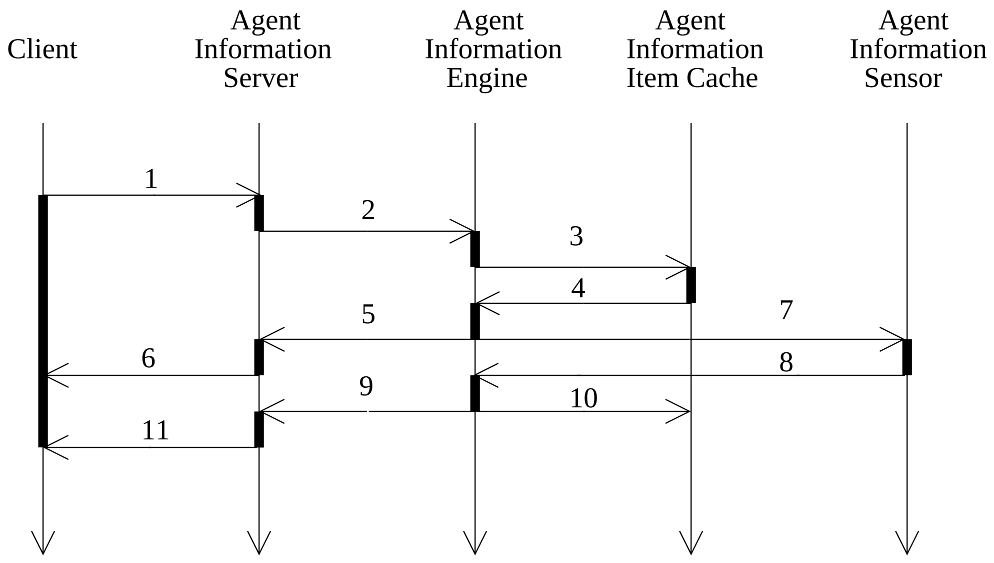


Figure 4.3: Process of information query on Information Agent

The process of information query is described as follows (see also Figure 4.3):

1. The client sends a query to the Information Agent;

2. On receiving the query, an Agent Information Server (thread) is created and processes the query, for example, checks the syntax of the query, maps the query to corresponding Agent Information Item/Sensor. The query is then forwarded to the Agent Information Engine;

3. The Agent Information Engine checks the Agent Information Item Cache to retrieve values for Agent Information Items;

4. The Agent Information Item Cache returns values of the query, if the queried information items are successfully located;

5. The Agent Information Engine returns values to the Agent Information Server which has forwarded the query;

6. The Agent Information Server returns information values to the client;

7. If the Agent Information Item has missed or the information item value is expired in the Agent Information Item Cache, the Agent Information Engine runs the corresponding Information Sensors to retrieve values of Agent Information Items;
8. The Agent Information Sensors return the latest values of information items to the Agent Information Engine;
9. The Agent Information Engine returns values to the Agent Information Server which has forwarded the query;
10. The Agent Information Engine also updates values in the Agent Information Item Cache;
11. The Agent Information Server returns values to the client.

4.3 WSRF-compliant Site Information Service

4.3.1 Introduction

The Site Information Service operates on the access point of a Compute Site (Computing Center). It receives queries from clients and returns resource information. It could also report resource information to higher level Grid aggregation information service.

The Site Information Service provides information of virtual machines for each Compute Site by communicating with Information Agents that run inside the virtual machines. The Site Information Service is WSRF compliant and implemented with Globus Toolkit 4 APIs and libraries. Users can query the Grid Information Service from standard Globus information service client. The Grid information service is hosted by Globus Toolkit service container and could be aggregated into Globus index service or Globus Toolkit Information Service Aggregator Framework [2].

The Site Information Service is constructed with following components:

- Site Information Server
  The Site Information Server receives query requests, checks the syntax, locates the query to Information Agents/Items, then forwards them to the Site Information Engine.
- Site Information Engine
  On retrieving query the Site Information Engine locates the information in the Site Information Cache. If the information is missed or expired, it queries on remote Information Agents for information update.
- Site Information Cache & Database
  The Site Information Item Cache is a block of memory which stores all the resource information. If the information is too large to load in memory, it is stored in a database.

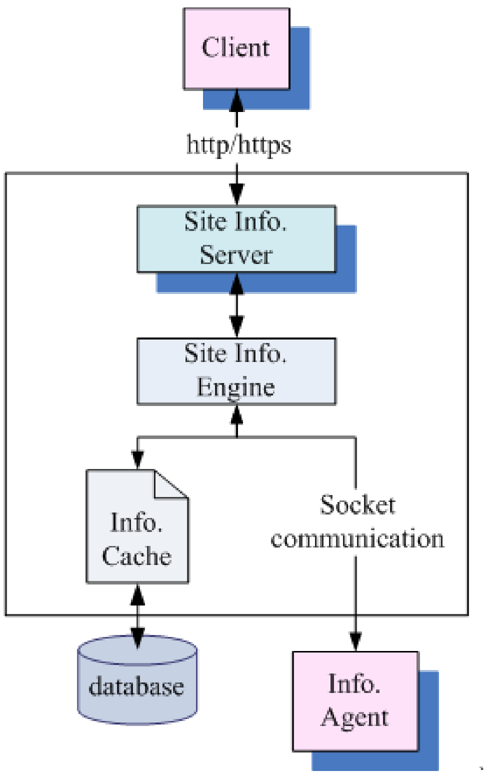


Figure 4.4: Site Information Service

4.3.2 The Resource approach: enable the stateful Site Information service

The Web service is stateless, which means that the Web service cannot remember the information or keep the state from one invocation to another. Globus Toolkit employ the *Resource* approach to make the Grid service stateful. A separate entity named *Resource*[1] is used to store all state information. A *Resource* contains a number of *Resource Properties* (RPs), which keep a set of states. In the Site Information service, the *Resource* contains the following *Resource Properties* such as *VMOSInfomation*, *VMMemoryInfomation*, *VMProcessorInfomation*, *VMSoftwareInfomation*, *VMDiskInfomation*.

A *Resource Home* is used to manage multiple *Resource*s. The implementation of *Resource Home* is based on the *findSingleton* method, which is called internally by the *ResourceContext* class provided by Globus Toolkit library:

```
    public Resource findSingleton() {
```

[1]The *Resource* which is in Italic font has different meaning from the compute resource mentioned in other parts. Section 2.1.1 introduces some related work of stateful Grid service and *Resource*.

```
        try {
            // Create a resource and initialize it.
            InfoResource infoResource = new InfoResource();
            infoResource.initialize();
            return infoResource;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

4.3.3 Site Information Service interface: WSRF-featured WSDL file

The *Resource Property* and the operations that manipulate on the Resource Property are defined in the interface – WSRF featured WSDL file. The *Resource Property* is the information retrieved from multiple virtual machines, which is defined as follows for example:

```
<xsd:element name="VMResourceProperty">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="tns:VMProcessorInfomation"
                    minOccurs="1"
                    maxOccurs="1"/>
            <xsd:element ref="tns:VMMemoryInfomation"
                    minOccurs="1"
                    maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="VMProcessorInformation">
    <xsd:complexType>
        <xsd:sequence>
        <xsd:elementname="processornumber"type="xsd:string"/>
        <xsd:elementname="processorspeed" type="xsd:string"/>
        <xsd:elementname="processorload" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="VMProcessorInformation">
</xsd:element>
```

The operation on *Resource Property* is defined as follows with appropriate messages:

```
<operation name="getVMResourceProperty">
    <inputmessage="tns:getVMResourcePropertyInputMessage"/>
    <outputmessage="tns:getVMResourcePropertyOutputMessage"/>
</operation>
```

The corresponding name space mapping is defined in the mapping file, which maps the language-neutral WSDL file to a set of stub classes implemented in Java language.

4.3.4 Implementation

When the Site Information Service is started, it firstly initializes the Resource Property and runs the Site Information Server. Then Site Information Server firstly makes queries on all the Information Agents that are listed in agent configuration file, which is shown as follows:

```
<resources>
    <agent addr="141.52.6.67" port="40001"/>
    <agent addr="141.52.6.36" port="40001"/>
</resources>
```

If the queried Information Agents are active, the Site Information Server then builds a map file in the memory and database, which record the mapping of Information Agents and the Site Information Items provided by the Information Agent. The Information Agents in the map file are identified by the IP address of virtual machine.

The Site Information Engine subscribes to all of the active Information Agents and the Information Agents periodically update the Site Information Engine with SOCKET messages, which are defined in following data type for example:

```
struct{
    string IP_address;
    string Processor_speed;
    string Processor_numer;
}
```

The updated resource information is loaded in the Site Information Item Cache. MySQL database is used for storing the information considering that the system performance could be damaged when too much information are cached in the memory. Three data tables are created in the database:

- *InfoAgent* table that defines Information Agents and IP addresses of the virtual machines where Information Agents run,
- *ClassAttributeMap* table that maps Information Agents to its Information Sensors, and

- *ClassAttribute* table that corresponds Information Sensor and its value.

Clients make queries on the Site Information Service by invoking Site Information Service operations to acquire *Resource Properties*. When queries arrive at the Site Information Service, it starts the Site Information Server to process the queried Information Items in the Resource Property. The Site Information Server checks the syntax, searches the related Information Agent/Information Sensor/Information Items in the mapping file. The Site Information Engine then queries on the Site Information Item Cache. If the information item is missed or expired, the corresponding Information Agents are executed and the values of Information Items are returned via SOCKET communication. Then the Site Information Server writes the values of Information Items, which are returned by Site Information Engine, to the variables of *Resource Property*. Standard mechanisms of WSRF-compliant service implementation can return *Resource Property* values to clients.

4.3.5 Deployment of Site Information Service in Globus Toolkit service container

The Site Information Service is deployed in the standard Globus Toolkit 4.0 service container. The deployment information is described in WSDD (Web Service Deployment Descriptor) file as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <service    name="metaservice/InformationService"
    provider="Handler" use="literal" style="document">
    <parameter name="className" value=
    "org.globus.metaservice.InformationService.impl.VMImpl"/>
    <wsdlFile>
share/schema/metaservice/InformationService/Info_service.wsdl
    </wsdlFile>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="handlerClass"
    value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="scope" value="Application"/>
    <parameter name="providers" value=
"GetRPProvider SubscribeProvider GetCurrentMessageProvider"/>
    <parameter name="loadOnStartup" value="true"/>
    </service>
</deployment>
```

Another deployment related file is JNDI deployment file which describes the *Resource Home* configuration. The JNDI deployment file is shown as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
    <service name="metaservice/InformationService">
        <resource name="home"
        type="org.globus.wsrf.impl.ServiceResourceHome">
            <resourceParams>
                <parameter>
                    <name>factory</name>
                    <value>
                    org.globus.wsrf.jndi.BeanFactory
                    </value>
                </parameter>
            </resourceParams>
        </resource>
    </service>
</jndiConfig>
```

Globus Toolkit provides tools to compile and deploy the Site Information Service in Globus service container with input of WSDD file and JNDI file.

4.4 Acquire resource information from VMware to Globus MDS

4.4.1 Introduction

Globus index service collects resource information to the central Virtual Organization (VO) information service. The Monitoring and Discovery System (MDS) is a group of Globus services to monitor and discover resources and services on computational Grids. Globus Toolkit offers a graphical interface, termed WebMDS, which could present resource information of MDS in the Web site.

This section considers a specific scenario: VMware ESX server is employed as VMM on the host resource, standard Globus Toolkit index service & MDS are used as Grid Information Service. VMware ESX server contains Pegasus CIMON, which provides programming interfaces to access resource information of virtual machines.

Thus an alternative solution is developed that the Information Provider of Globus Information Service communicates with Pegasus CIMON via the HTTP/HTTPS protocol and retrieves resource information of virtual machines to the Globus information service (Figure 4.5).

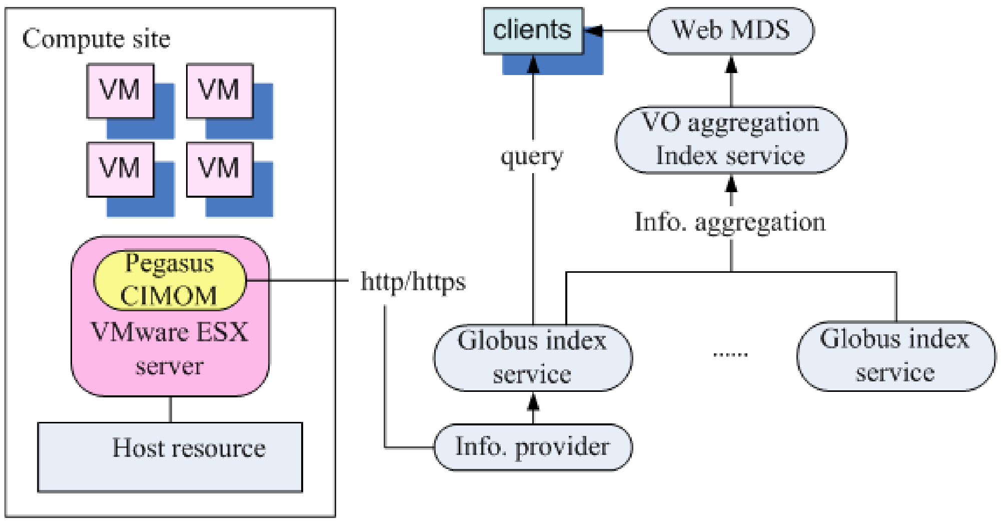


Figure 4.5: Acquire resource information from VMware to Globus MDS

4.4.2 CIM environment of VMware ESX Server

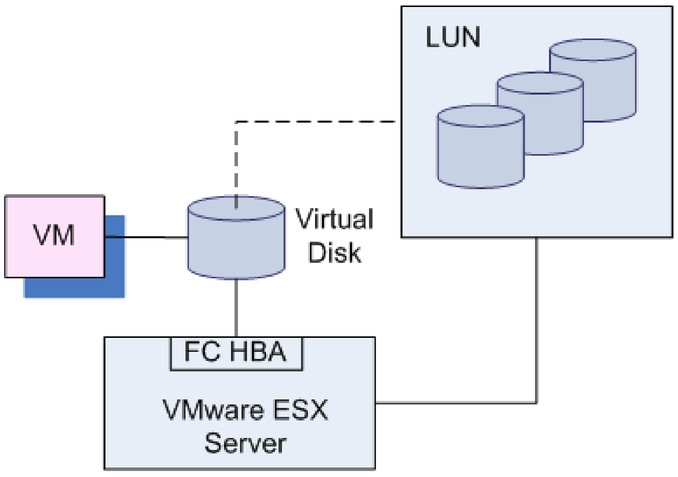


Figure 4.6: Sample environment of VMware ESX server

VMware ESX server is a commercial virtualization product of VMware Inc. Common Information Model (CIM) [129] is defined as an international standard by the Distributed Management Task Force (DMTF) [130]. The VMware ESX server together with CIM SDK provides a CIM-compliant object model for virtual machines and related storage devices. Figure 4.6 shows a typical configuration environment for VMware ESX server. The virtual machine contains a virtual disk, which resides as a virtual disk file on a storage array, e.g., Logical Unit Number (LUN).

4.4.3 Information provider for Globus MDS

The information provider runs on the access point of Computer Site. It accesses various VMware ESX servers, retrieves resource information of virtual machines, translates the information into GLUE schema, and finally provides the information to the Globus MDS back end. This section discusses the information retrieval process. Section 4.5 introduces information translation from CIM schema to GLUE schema and Section 4.4.4 presents the Globus MDS back end extension to obtain resource information from the information provider.

VMware ESX server provides the CIM SDK, which helps CIM client (information provider of Globus MDS) to access VMware ESX server and retrieve virtual machine information.

The CIM SDK is composed of the following elements:

- Pegasus CIMOM (CIM Object Manager) [158] to allow clients to explore virtual machines and related allocated resources on ESX Server hosts.
- MOF (Managed Object Format) files specifying the CIM SDK schema.

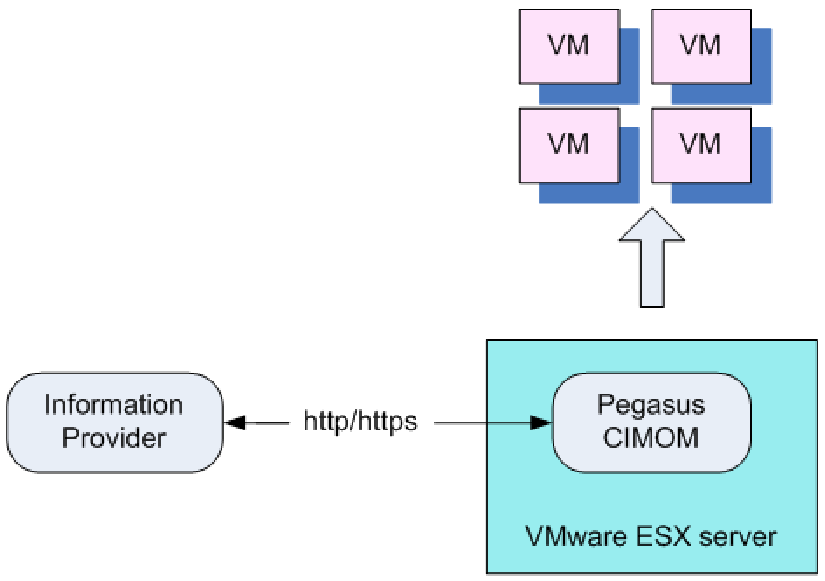


Figure 4.7: Retrieve information from CIMON of VMware ESX server

The information provider is programmed with Java to remotely invoke CIM SDK APIs via HTTP/HTTPs communication. After the resource information is retrieved, the information provider thus translates it from the CIM schema to the GLUE schema, then submits it to the Globus MDS back end. The detailed information translation is discussed in Section 4.5.

4.4.4 Globus MDS extension

The information provider now retrieves resource information from virtual machines then provides resource information to the back end of Globus index service in following steps:

- Enable the provider
  Demands in the configuration file invoke to map the name of information provider and the executable in the deployment file of Globus index service. In our implementation the information provider's name is *vminfo*, which communicates with VMware ESX server and performs the information translation.

  It maps to the *VMInformationProvider* in the configuration.

  ```
  <parameter>
     <name>executableMappings</name>
        <value>
           VMInformationProvider=vminfo
        </value>
  </parameter>
  ```

  It is also required to move the information provider executable to the default location of Globus toolkit installation. In the implementation, it is

  ```
  $GLOBUS_LOCATION/libexec/aggrexec/
  ```

  Globus index service can update the information by periodically executing the information provider or pulling the latest information from the information provider on query.

- Register the information provider to Globus index service
  To register the information provider to Globus index service, a registration file is needed. In the implementation, the registration file for the virtual machine information provider is shown as below:

  ```
  <ServiceGroupRegistrationParameters
      xmlns="http://mds.globus.org/servicegroup/client">
      <RefreshIntervalSecs>600</RefreshIntervalSecs>
      <Content xsi:type="agg:AggregatorContent"
          xmlns:agg="http://mds.globus.org/aggregator/types">
      <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
      <agg:ExecutionPollType>
  ```

```
    <agg:PollIntervalMillis>300000</agg:PollIntervalMillis>
    <agg:ProbeName>VMInformationProvider</agg:ProbeName>
    </agg:ExecutionPollType>
    </agg:AggregatorConfig>
    <agg:AggregatorData/>
    </Content>
</ServiceGroupRegistrationParameters>
```

With standard Globus Toolkit MDS command, the information provider can be registered to the Globus index service, for instance,

```
mds-servicegroup-add -s https://141.52.6.36:8443 \
/wsrf/services/DefaultIndexServiceregister.xml
```

At the current stage, users can query Globus index service to retrieve virtual machine information with standard Globus Toolkit client tools and commands.

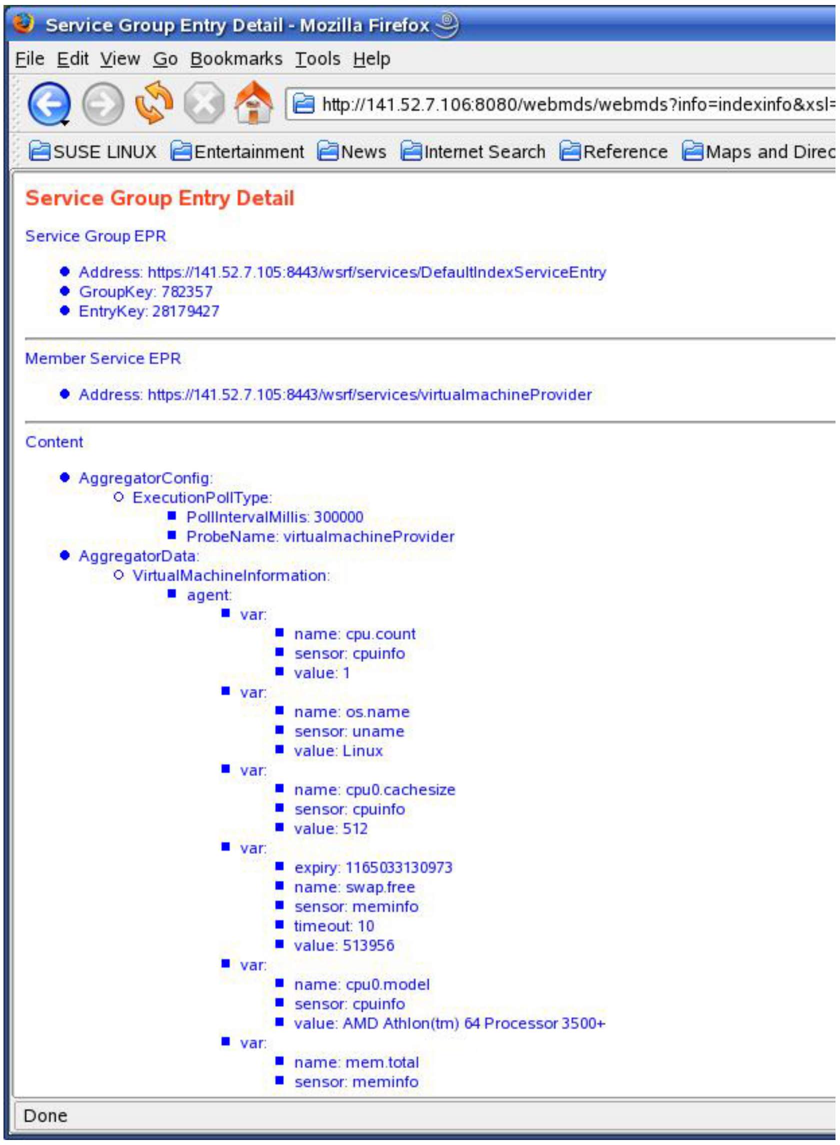
To build Globus central Virtual Organization (VO) information service with underlying Globus index service, following steps are performed:

- Register Globus index service to VO aggregation information service
  In the host where Grid index service operates, a configuration file should be built. Thus identifies the VO aggregation information service to which the Grid index service wants to register. An example of the configuration file for Globus index service is shown as follows:

  ```
  <config>
      <upstream>
  https://141.52.7.106:8443/wsrf/services/DefaultIndexService
      </upstream>
  </config>
  ```

  On the host where the central VO information service resides, a configuration file should be built to identify the low level Grid index services that want to register to the central VO information service. An example configuration is shown as follows:

  ```
  <config>
      <downstream>
  https://141.52.6.36:8443/wsrf/services/DefaultIndexService
      </downstream>
  </config>
  ```


Service Group Entry Detail - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://141.52.7.106:8080/webmds/webmds?info=indexinfo&xsl=
SUSE LINUX
Entertainment
News
Internet Search
Reference
Maps and Direc
Done

Service Group Entry Detail

Service Group EPR

- Address: https://141.52.7.105:8443/wsrf/services/DefaultIndexServiceEntry
- GroupKey: 782357
- EntryKey: 28179427

Member Service EPR

- Address: https://141.52.7.105:8443/wsrf/services/virtualmachineProvider

Content

- AggregatorConfig:
  - ExecutionPollType:
    - PollIntervalMillis: 300000
    - ProbeName: virtualmachineProvider
- AggregatorData:
  - VirtualMachineInformation:
    - agent:
      - var:
        - name: cpu.count
        - sensor: cpuinfo
        - value: 1
      - var:
        - name: os.name
        - sensor: uname
        - value: Linux
      - var:
        - name: cpu0.cachesize
        - sensor: cpuinfo
        - value: 512
      - var:
        - expiry: 1165033130973
        - name: swap.free
        - sensor: meminfo
        - timeout: 10
        - value: 513956
      - var:
        - name: cpu0.model
        - sensor: cpuinfo
        - value: AMD Athlon(tm) 64 Processor 3500+
      - var:
        - name: mem.total
        - sensor: meminfo

Figure 4.8: Snapshot of WebMDS

- Build WebMDS on central VO information service
  WebMDS requires that the Globus services are deployed in Tomcat container instead of Globus default container. Configuration is also required for the index services that WebMDS plans to monitor. At the stage, users can query virtual machine information from central VO information service or WebMDS. An example result is shown in Figure 4.8.

4.5 From CIM schema to GLUE schema: information translation

4.5.1 CIM schema for virtual machine information

The CIM (Common Information Model) schema [129], which is a standard created by DMTF (Distributed Management Task Force), provides a common definition for management information of systems, networks, applications and services. CIM is a conceptual information model for describing computing and business entities in enterprize environments. The fundamental goals of CIM are common definitions that enable vendors to exchange semantically rich management information between a wide variety of systems.

The CIM standards are used by the Storage Networking Industry Association (SNIA) as the basis for standards profiles specific to certain areas of storage management. These profiles specify how the CIM should be applied to a specific kind of storage management. VMware has chosen to implement the IBVP (In-band Virtualizer Profile) because it is well adapted to model VMwares storage virtualization technology.

The SMI-S (Storage Management Initiative Specification) schema for the sample VMware ESX server environment is shown using UML in Figure 4.9. *ESXComputerSystem* is the kernel object of the system. It associates *VM*, *VirtualDisk* and *FC HBA&LUN* (Fibre Channel Host Bus Adaptor & Logical Unit Number) with *HostedDependency*, *HostedStoragePool* and *SystemDevice* relationships respectively. *VM* is associated with *VirtualDisk* with *ArchiConnection* relationship. The latter is associated with *FC HBA&LUN* in *ConcreteComponent* relationship.

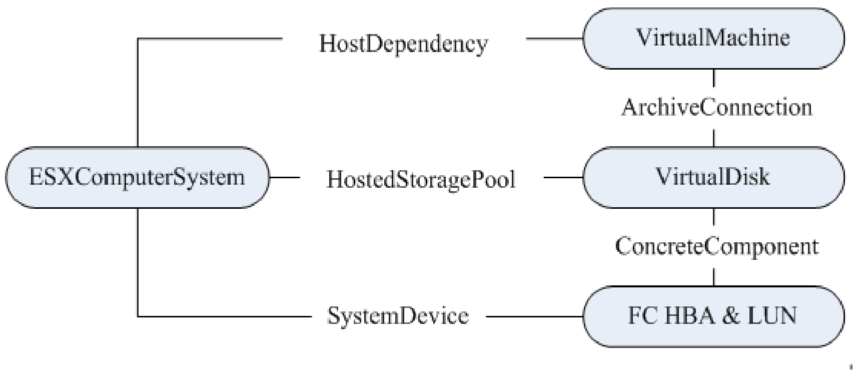


Figure 4.9: CIM schema for VMware ESX server

4.5.2 GLUE schema

The GLUE (Grid Laboratory Uniform Environment) schema [136] is an abstract model for Grid resources and mappings to concrete schemas which could be used by information services within computational Grids. The essential idea of the GLUE schema is to provide an information model that can be used to exchange pieces of information among different knowledge domains and Virtual Organizations [49]. The GLUE schema is widely used in production Grids such as EGEE [35], OSG [154], NorduGrid [97], and APAC Grid [83].

The GLUE schema is defined in UML diagrams (1.3 version) or XML (1.2 version). The GLUE schema defines so called core entities, such as *Site*, *Service*, *ComputingElement* and *StorageElement*. The relations between core entities are represented in concept level, such as objects and properties (see also Figure 4.10).

4.5.3 Information translation

The virtual machine information of VMware is organized with CIM schema. In production computational Grids, resource information is defined by GLUE schema. The information provider therefore needs to translate the CIM information into GLUE information, then marshes the data into GLUE XML file.

There are two main technical problems when translating CIM information to GLUE information:

- The GLUE schema does not define the virtual machine concept and its related objects.
- The CIM schema and the GLUE schema make their own representation at different levels. The CIM schema is represented with various Java/C++ objects and the GLUE schema is defined in XML and UML entities. It is thus impossible to make a direct mapping from CIM schema to GLUE schema.

We solve the above problems as follows:

- The GLUE schema has been extended accordingly to support virtual machine concept. A *VirtualMachine* class is created by inheriting the *Host* class in the GLUE schema. Attributes and operations are extended in *VirtualMachine* classes to represent virtual machine concepts. Each *Host* is associated with multiple *FCHBA* classes and *StoragePool* classes. Figure 4.11 shows the extended GLUE schema.
- The GLUE schema is represented in various high level formats, such as XML schema and UML class diagram. The CIM schema, on the other hand, provides the object oriented format. The CIM schema shipped by VMware is represented in Java objects. Thus there is no direct mapping from the CIM schema to the GLUE schema. To solve this problem, extended GLUE schema generates a set of binding Java classes with JAXB binding

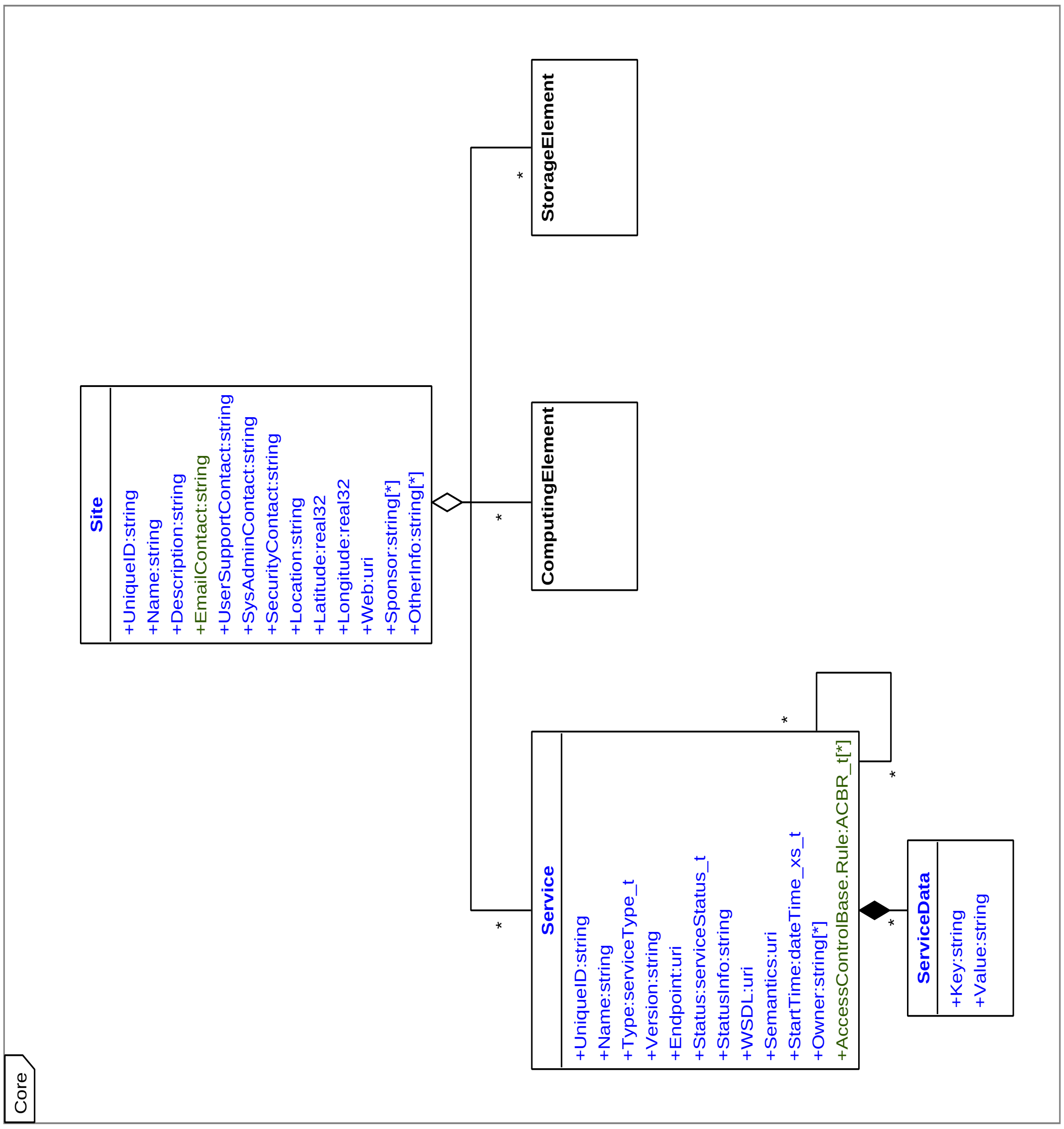


Figure 4.10: Overview of GLUE schema

compiler. The Java Architecture for XML Binding (JAXB) [37] provides a fast and convenient way to bind an XML schema to Java representations. With the JAXB compiler and binding tools, the Java binding classes could be generated from GLUE schema. Now the GLUE binding Java classes can map to the CIM Java classes by invoking operations of CIM Java classes.

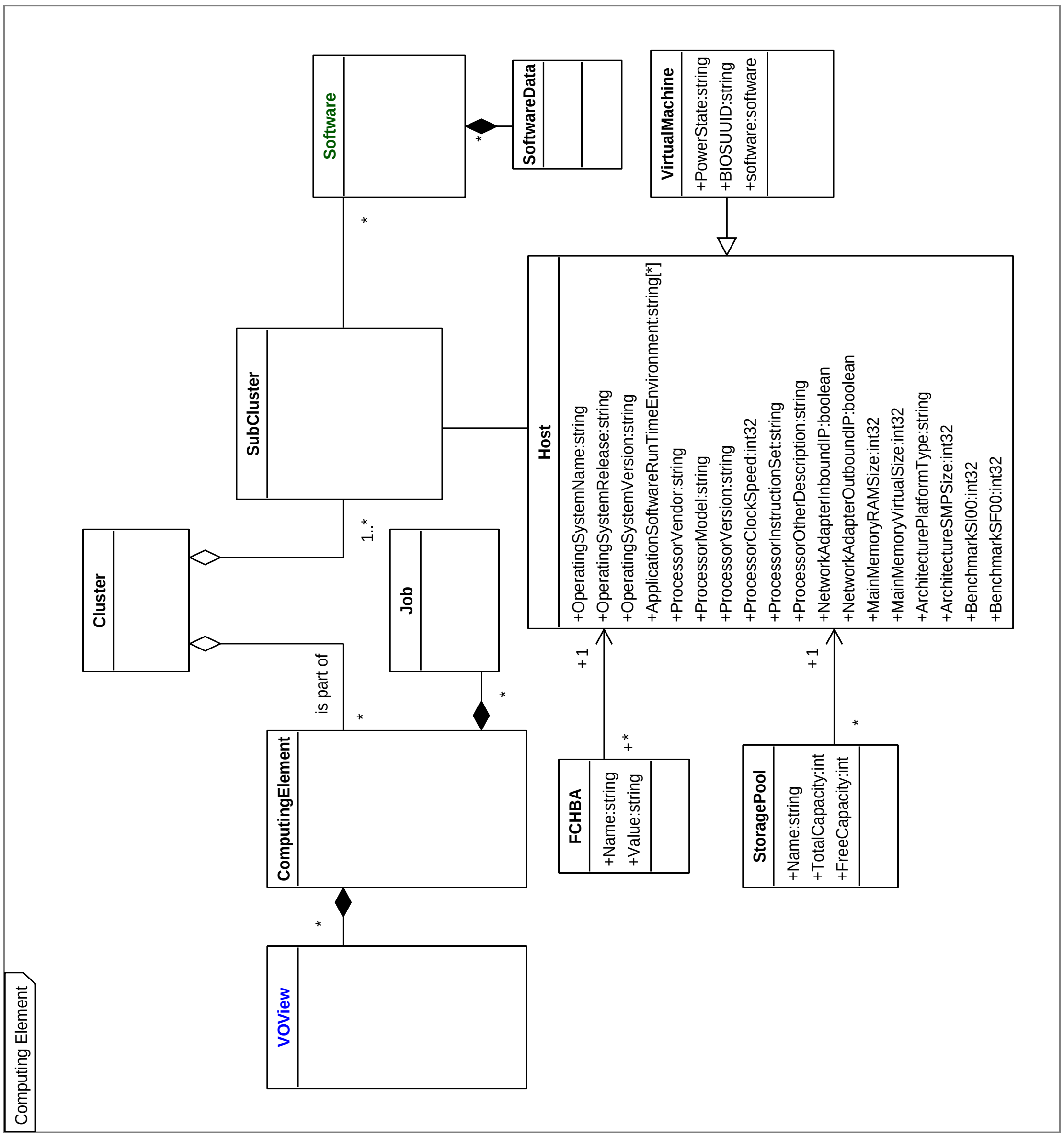


Figure 4.11: GLUE schema with virtual machine extension

4.6 Summary

This chapter presents the Information Service, which retrieves distributed virtual machine information to the Grid level. A WSRF-compliant Site Information Service is developed and communicates with distributed lightweight Information Agents, which run inside virtual machines. A typical computing environment is investigated that VMware ESX server and Globus

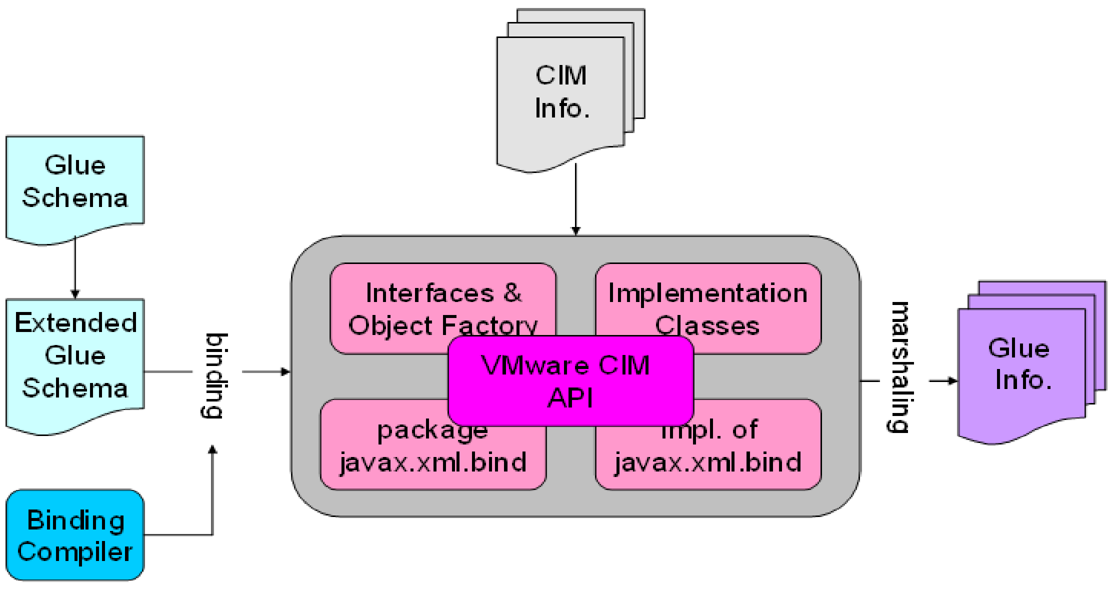


Figure 4.12: Translation from CIM schema to GLUE schema

Toolkit MDS are deployed. The specific solution for this scenario is discussed: retrieving resource information by communicating VMware CIMOM and providing information to the back end of Globus index service. In the chapter another important topic is also discussed: translating CIM based virtual machine information to Grid information defined by GLUE schema.

Chapter 5

A New Virtualization Service for Operating Virtual Environments

5.1 Overview

The Virtualization Service is a software layer on distributed host resources. The Virtualization Service offers on-demand provision of virtual machines and VDEs. The Virtualization Service is implemented in the distributed and hierarchical favors with standard Web service. Current implementation of the Virtualization Service can operate on popular virtual machine hypervisors, such as Xen server and VMware ESX server. The Virtualization Service is designed to work on the target Grid system which is defined in Section 3.4.3.

The Virtualization Service contains the following components:

- Site Virtualization Service
  The Site Virtualization Service resides on the access point of a computing center and supports the functions of virtual machine or VDE provision from the host resources inside the computing center. The Site Virtualization Service controls multiple underlying host resources by communicating with Virtualization Agents.

- Virtualization Agent
  A Virtualization Agent exists on each host resource (VA), which gets commands from the Site Virtualization Service and operates on virtual machines that it manages.

- Database for Management Information & Software Stack
  Databases are required to store the Management Information, such as user management policies, resource allocation status for users, virtual machine information, and Software Stack (SS), for example, virtual machine disk images, software packages and libraries.

The Site Virtualization Service offers the following functions:

- VM Requirement
  - create a new VM, and
  - require an existing VM;
- VM Operation
  - start/Shutdown/suspend a VM,
  - clone a VM,
  - run a script/command/executable in a VM, and
  - copy files from/to a VM.
- VDE function
  Users of Virtualization Service could build VDE functions by invoking various VM functions, e.g., VDE requirement, VDE start/shutdown/migration and VDE configuration.

VDE functions should be implemented at the user level and adapted to specific application domains since different Grid user group might demand different Grid computing environments. Virtualization service here takes a role of “enabling technology” and makes it possible for Grid users to create a customized computing environment.

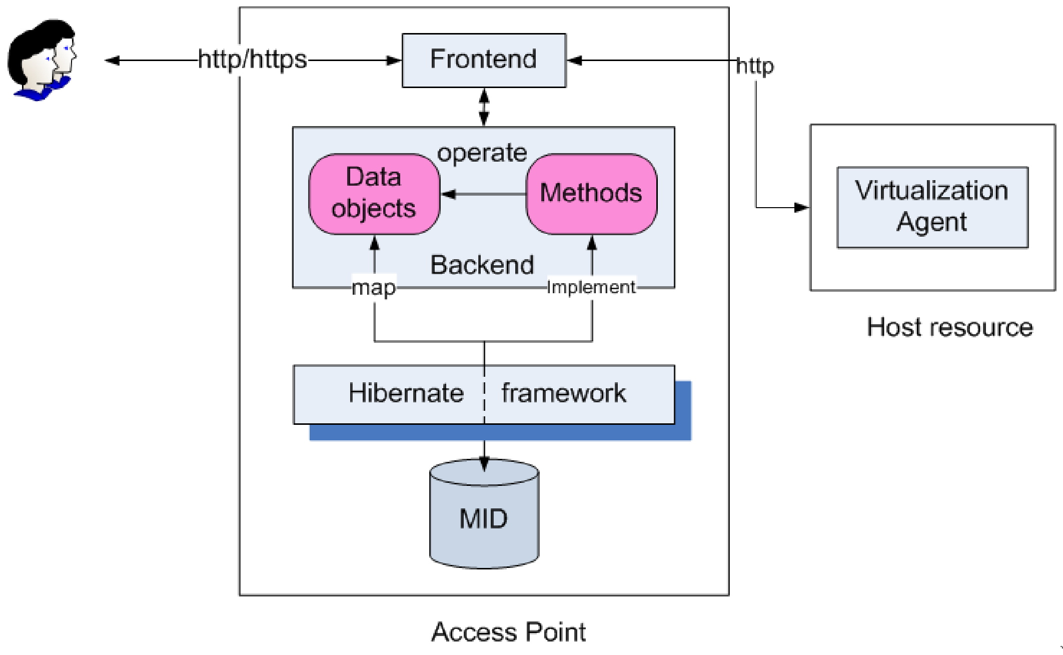


Figure 5.1: Site Virtualization Service

5.2 Site Virtualization Service

5.2.1 Introduction

The Site Virtualization Service consists of two components:

- Front-end Site Virtualization Service
  The Front-end Site Virtualization Service is responsible for the Site Virtualization Service business logic. It accepts requirements from users and contacts the underlying Virtualization Agents for virtual machine operation.

- Back-end Site Virtualization Service
  The Back-end Site Virtualization Service needs to access the Management Information Database for virtual machines manipulation. The Back-end Service is therefore constructed to help the Front-end Site Virtualization Service to access the Management Information Database.

The Management Information Database stores the following information:

- management policies for users to access virtual machines inside the computing center.
- current virtual machine allocation for users, e.g., virtual machine ID and duration,
- virtual machine information, such as virtual machine profile and state,
- underlying Virtualization Agents which have registered themselves on the Front-end Virtualization Service.

5.2.2 Virtual machine profile

Operations on virtual machines need a parameter, *VMProfile*, to describe virtual machine properties, e.g., IP address, CPU bandwidth and software installed. The *VMProfile* points to an XML file defined by the extended GLUE schema, which is introduced in Section 4.5.3. *VMProfile* extends the *Host* type in GLUE schema by defining a new data type *ServiceType* to record application level software installed in the virtual machine:

```
<xs:complexType name="ServiceType">
<xs:sequence>
<xs:element name="Name" type="xs:string" minOccurs="0" />
<xs:element name="Type" type="ServiceTypeOpenEnum"
minOccurs="0"/>
<xs:element name="Version" type="xs:string" minOccurs="0"/>
```

```
<xs:element name="Endpoint" type="xs:anyURI" minOccurs="0"/>
<xs:element name="Status" type="ServiceStatusEnum"
minOccurs="0"/>
<xs:element name="StatusInfo" type="xs:string"
minOccurs="0"/>
<xs:element name="WSDL" type="xs:anyURI" minOccurs="0"/>
<xs:element name="Semantics" type="xs:anyURI"
minOccurs="0"/>
<xs:element name="StartTime" type="xs:dateTime"
minOccurs="0"/>
<xs:element name="Owner" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="Data" minOccurs="0"
maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="Key" type="xs:string"/>
<xs:element name="Value" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="UniqueID" type="UniqueIDType"
use="required"/>
</xs:complexType>
```

To access a virtual machine users need a *UserID* and a *password*, which are stored internally as a data structure *accessData* by the Virtualization Service.

5.2.3 The Front-end Site Virtualization Service

The Front-end Site Virtualization Service provides two types of functions: virtual machine request/release and virtual machine management, which are implemented in the asynchronous flavor. When clients invoke operations of Virtualization Service, an object *Job* is built internally to record the operation on virtual machine, then a *gveJobState* message is returned to the clients, which contains the *uniqueJobID* for later reference.

```
<xs:complexType name="gveJobState">
    <xs:sequence>
        <xs:element name="errorDescription"
        type="xs:string" minOccurs="0"/>
        <xs:element name="jobState"
        type="tns:jobStateType" minOccurs="0"/>
        <xs:element name="uniqueJobID"
        type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="jobStateType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="SUCCESS"/>
        <xs:enumeration value="RUNNING"/>
        <xs:enumeration value="QUEUED"/>
        <xs:enumeration value="ERROR"/>
    </xs:restriction>
</xs:simpleType>
```

The functions are shown as follows:

- Virtual machine request/release
  - requestVM
    Users require a virtual machine firstly by invoking the *requestVM* operation with the parameters: *accessData* and *VMProfile*. Users then get the *uniqueJobID* for later reference.
  - queryVM
    Users can query the result of the *requestVM* operation by calling the *queryVM* operation with the parameter *uniqueJobID*. The output of the *queryVM* operation is the *gveJobState*.
  - requstVMResult
    When the *gveJobState* returned from the *queryVM* operation is *SUCCESS*, then the user can call the *requstVMResult* operation to get the virtual machine access, e.g., IP address in *VMProfile*.
  - releaseVM
    Users release the virtual machine, which is acquired by above steps, by invoking the *releaseVM* operation with the parameter *accessData* of the virtual machine.
- Virtual machine management:
  - manageVM
    Users operate on virtual machines by invoking the *manageVM* operation, which takes two parameters as input: virtual machine *ID* and *action* on virtual machines.

    The *action* could be, for example:
    - start/shutdown/suspend a virtual machine,

        - ■ clone a virtual machine, and
        - ■ run scripts in a virtual machine.
    - queryManageVMJobState
      Users then get the result of the *manageVM* operation by invoking the *queryManageVMJobState* operation.

5.2.4 The Back-end Site Virtualization Service

Functions offered by the Front-end Site Virtualization Service normally demand access to some management information, which is implemented in the Back-end Site Virtualization Service.

The implementation contains following logical steps (see also Figure 5.1):

- The Back-end Site Virtualization Service defines a set of data objects where management information is stored, such as user information and Virtualization Agent information.
- The Back-end Service defines methods that operate on the data objects, for example, *create*, *update*, *delete*.
- Hibernate object relational mapping framework [141] automatically implements the methods of data object operations.
- Hibernate framework generates a database, maps each data object defined above to a SQL table in the database. Hibernate framework is discussed in Section 5.2.5.

The Management Information Database (MID) contains following SQL tables:

- AgentJobEntity
  The *AgentJobEntity* represents Virtualization Agent information and its related job. It contains:
    - Virtualization Agent URL,
    - Job ID, and
    - User ID that runs the job on the Virtualization Agent.
- VMEntity
  VMEntity stores the virtual machine information such as:
    - VM ID,
    - VM owner,
    - Virtualization Agent URL,
    - Timestamp,

  - VM profile, which is represented in GLUE schema.

- JobEntity
  The *JobEntity* stores job information, such as:
  - Job owner,
  - Job stat,
  - Job content, e.g., operation, parameter.
- UserEntity
  The *UserEntity* provides user information, for instance:
  - *AccessData*, which includes a user ID and password;
  - *UserResourceAccount*, such as resources that the user has consumed.
- VMRequestEntity
  The *VMRequestResultEntity* associates a virtual machine request and the virtual machine reserved with a user. The *VMRequestResultEntity* is used when a Virtualization Service client wants to acquire a virtual machine but only has a Job ID. It maps the Job ID to the unique identifier of the virtual machine reserved for a user.

5.2.5 Access the Management Information Database via Hibernate framework

Hibernate object-relational mapping library [141] for the Java language provides a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate framework helps build a database and connection from the application programs. It works in four steps:

1. Hibernate maps the data structures of application programs to database tables with a mapping file. The following example shows the mapping file, which maps the data structure *JobEntity* to *JOBENTITY* table in the database:

```
<Hibernate-mapping package="org.gve.inf.serv.dataprov">
    <class name="JobEntity" table="JOBENTITY" >
        <id name="id" column="ID">
            <generator class="native"/>
        </id>
        <property name="jobs_owner" not-null="true"/>
        <property name="jobs_jobState"
        type="serializable" not-null="true"/>
    </class>
</Hibernate-mapping>
```

2. Hibernate reads the user defined configuration file to determine how to connect the database. In the configuration file, users define the parameters to connect the database, such as:
   - the database URL in which the objects are persisted,
   - the JDBC driver that handles the connection to the specified database,
   - the SQL dialect used by the database,
   - credential details used to access the database, and
   - a list of all Hibernate objects relational mapping files.
3. Then Hibernate creates a session configuration class to perform connection to the database.
4. Finally Hibernate implements the methods to access data structures stored in the database.

5.3 Virtualization Agent

5.3.1 Introduction

The Virtualization Agent (VA) is a Web service which runs on host resources. It receives operation commands from the Site Virtualization Service and communicates with the specific underlying VMM, e.g., VMware ESX server and Xen server, which are installed on host resources. The Virtualization Agent is thus VMM dependent. In other words, for each type of VMM, a corresponding Virtualization Agent should be implemented. In the thesis, two types of Virtualization Agents are implemented for VMware ESX server and Xen server respectively. Software Stack database is required when the Virtualization Agent demands the VMM to create new virtual machines. The software stack database contains the following information:

- virtual machine disk images used when the virtual machine is created, and
- various of software packages to build pre-configured virtual machines.

5.3.2 Internal operational states of virtual machine

A virtual machine could be marked with three states: *Free*, *Reserved* and *Acquired*. The *Free* state means that no user is currently accessing the virtual machine. When a user requests a virtual machine, the virtual machine is marked as *Reserved*, which means that the virtual machine is allocated to a user and waits for further operations. The user receives a confirmation from the Virtualization Service, then it starts continued operations on the virtual machine, e.g., load operating system and run scripts in the virtual machine. This state is marked as *Acquired*.

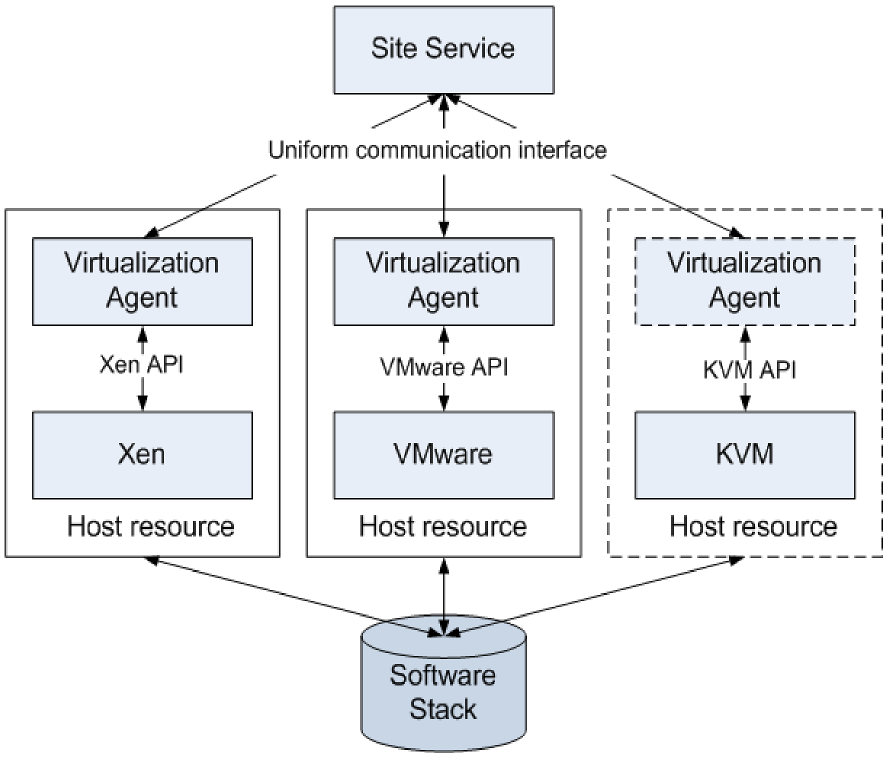


Figure 5.2: Virtualization Agent

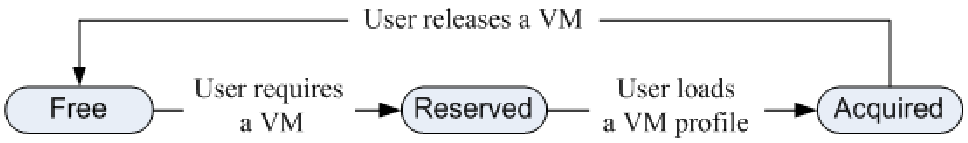


Figure 5.3: States of virtual machine

5.3.3 Interface of Virtualization Agent

For the Virtualization Agent Web service, WSDL file defines asynchronous messages and bindings, which are similar with the Site Virtualization Service. The Virtualization Agent operations are implemented in the asynchronous pattern. Virtualization Agent offers following operations:

- reserveVM
  The *reserveVM* function marks a virtual machine with the state *Reserved*. It takes the virtual machine ID as the input parameter and returns a boolean value to indicate whether the operation is successful.

- acquireVM

The *acquireVM* function marks a virtual machine with the state *Acquired*. It takes the virtual machine ID as the input parameter and returns the virtual machine profile, *VM-Profile*.

- releaseVM
  The *releaseVM* function marks a virtual machine with the state *Free*. It takes the virtual machine ID as input parameter and returns a boolean value to indicate whether the operation is successful.

- copyFileToVM
  The *copyFileToVM* function copies files to the virtual machine from the host resource. The input parameter is virtual machine ID, which is used as a template for the clone operation, and the filename. It returns a boolean value to indicate whether the operation is successful.

- runScriptInVM
  The *runScriptInVM* function runs executables inside virtual machine. Its input parameters are virtual machine ID and the filename of the executable. It returns a boolean value to indicate whether the operation is successful.

- cloneVM
  The *cloneVM* clones a virtual machine. Its input parameters are the virtual machine ID and outputs a cloned virtual machine profile, *VMprofile*.

- startVM/shutdownVM/suspendVM
  Manage functions like *startVM*, *shutdownVM*, *suspendVM* operate on virtual machines with virtual machine ID as input parameter, and returns a boolean value to indicate whether the operation is successful.

- findVM/findVMTemplate
  The *findVM/findVMTemplate* searches all of the virtual machines or the virtual machine templates on the host resource and return the desired virtual machine or virtual machine template.

5.3.4 Implementation

The JAXB framework can generate code skeletons from the WSDL file of Virtualization Agent Web service. The code skeletons define the data structures and functions for virtual machine operations.

The virtual machine functions should be implemented with underlying VMM APIs. Functions of *startVM/shutdownVM/suspendVM*, *copyFileToVM*, *runScriptInVM*, *cloneVM*, *findVM* and *findVMTemplate* are easily implemented since direct Xen/VMware APIs are invoked. The *reserveVM* calls the *findVM/findVMTemplate* functions then marks the returned virtual machine as “reserved”. The functions of *acquireVM* and *releaseVM* are implemented by changing the state of virtual machine to states of *Acquired* or *Free*.

5.4 Deployment of Virtualization Service

To be able to deploy various components (Site Virtualization Service and Virtualization Agent), a Graphical User Interface is developed to manage the Grid Virtualization Service. The main function of the GUI is to register Virtualization Agents on the Site Virtualization Service and to display the virtual machines allocated to users.

To deploy the Virtualization Service on distributed host resources the following steps are required for a Grid Virtualization Service instance:

1. deploy the Site Virtualization Service instance;
2. configure the identity of the Site Virtualization Service, so that Virtualization Agents can register;
3. create or register user accounts in the User Information Database;
4. register one Virtualization Agent to Site Virtualization Service.

5.5 Summary

This chapter discusses the design and the implementation of Virtualization Service. The Virtualization Service provides standard Web service interfaces for users to create, manage and deploy virtual machines and Virtual Distributed Environments on Grid infrastructures.

The Virtualization Service distinguishse itself from related work [1, 106, 110] in that:

- The Virtualization Service is designed and implemented in a modularity manner and the system components are wrapped with standard Web service interfaces. The modular design philosophy offers advantages such as scalability, availability and interoperability to the system.
- The Virtualization Service is designed and implemented in the hierarchical flavor. The higher level service, Virtualization Site Service, provides a general interface, which is virtual machine technology independent; the lower level service, Virtualization Agent, handles virtual machine specific implementations. The hierarchical design pattern makes the system more scalable to allow incorporation of new virtual machine technologies.

Chapter 6

Grid Workflow in a Box: a Novel Method for Workflow Organization

6.1 Introduction

This chapter presents details of how to construct a VDE: Grid Workflow in a Virtual Box (GWVB), which consists:

- a set of virtual machine disk images,
- software packages & libraries,
- and several installation/configuration/control scripts.

Grid Workflow in a Virtual Box is characterized as a system which contains data and the scripts which operate on the data. The usage methodology benefits users by offering automatic installation sources and processes to build a Grid workflow system on desired computing resources. This use scenario justifies the usage and management of a VDE – submission, execution and migration of a GWVB is with simple control.

6.2 Virtual Data System: a Grid Workflow System

6.2.1 Introduction

Grid workflow is a popular system for Grid application organization, which was introduced in Section 2.1.3. The Chimera Virtual Data System (VDS) is a Grid workflow system, which provides a set of tools for expressing, executing and tracking the results of workflows [169]. The

VDS provides a means to describe a desired data product and to produce it in the Grid environment. The VDS provides a catalog to describe a set of application programs (*transformations*) , and then track all the data files (*derivations*) produced by executing those applications.

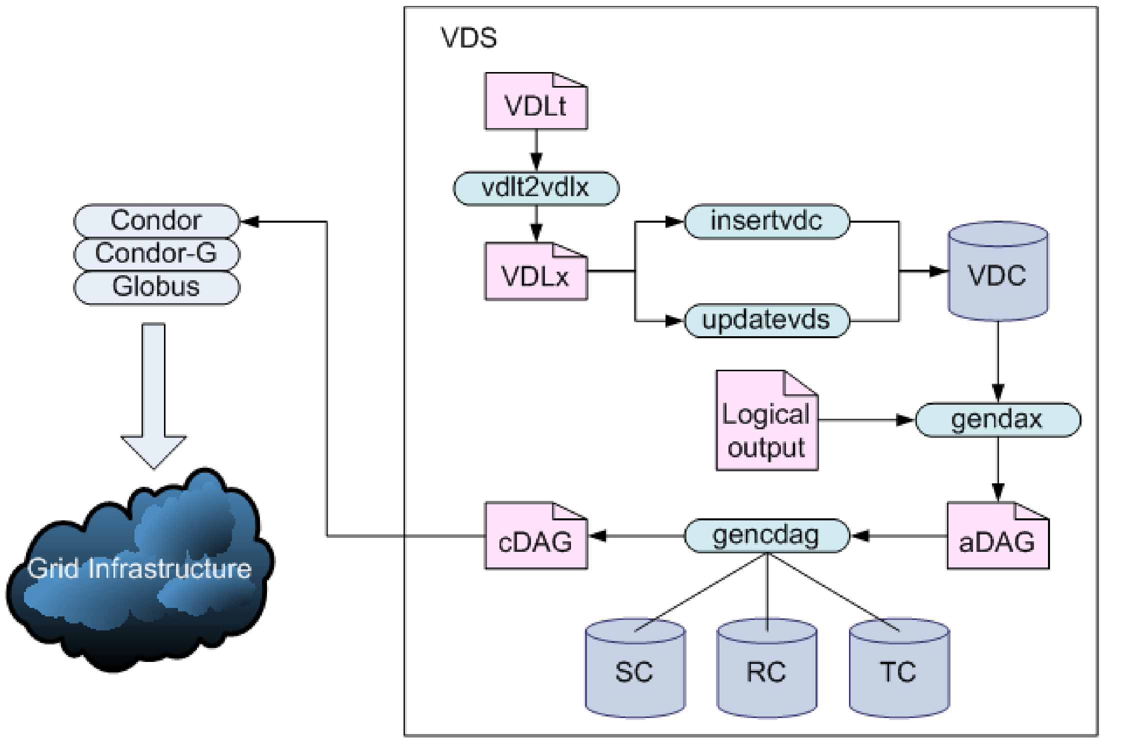


Figure 6.1: Virtual Data System

Figure 6.1 provides an overview of the VDS. Users define the workflow with the Virtual Data Language (VDL) in a VDL text (VDLt) file. The system tool *vdlt2vdlx* converts the VDLt file into VDL XML (VDLx) file. A Virtual Data Catalog (VDC) contains virtual data definitions – the data files generated as output files by some programs and needed by some programs as input files. The Virtual Data Catalog can be changed by the tools of *insertvdc* and *updatevdc* from VDLx files. Then the abstract DAG (aDAG) file, which describes the task and data in the workflow with abstract definitions, is produced by the tool *gendax*. A concrete DAG (cDAG) file defines the workflow with real executables and data files and can be submitted to Grid resources for execution. To convert the abstract DAG files to concrete DAG files by the VDS tool *gencdag*, the following information is required:

- Site Catalog
  Site Catalog (SC) records Compute Site profiles on the Grids, such as *site name*, *gatekeeper*, and *gridftp* server. With the information from the Site Catalog, *gencdag* selects proper sites to execute the workflow.

- Transform Catalog
  Transform Catalog (TC) stores the mapping of logical application names to physical executables. *gencdag* translates the logical application names in abstract DAG to real executables in concrete DAG.

- Replica Catalog
  Replica Catalog (RC) retains the information that maps logical data files to physical data files. *gencdag* converts the logical data files in abstract DAG to physical data fiels in concrete DAG.

When the concrete workflow is generated, it is then submitted to computational Grids via Condor-G [44] and Globus Toolkit [135].

6.2.2 Grid model for running VDS

The Grid system where the VDS workflow is executed, contains the following elements with different functionalities (see Figure 6.2):

- Catalog Host
  The Catalog Host (CH) contains VDC, VDS packages and DAG directories, where the VDS workflow and VDC are prepared and manipulated.

- Submit Host
  The Submit Host (SH) is installed with Globus Toolkit, personal Condor and Condor-G, via which the concrete DAG are submitted to the Grids. In our implementation, Catalog Host is merged to the same server of Submit Host.

- Compute Site
  The computational Grid is composed of multiple Compute Sites (CS). In the Compute Site, Globus gatekeeper is installed in the head node and can accept VDS workflows. The Compute Site contains a resource broker, for example Condor [44], PBS [156] and LSF [142].

6.3 Constructing a virtual Grid infrastructure for VDS

6.3.1 Introduction

This section discusses how to use the GVE to automatically build VDS system on distributed virtual machines. The process of building VDS contains several stages by automatically running and GVE clients and installation scripts by calling GVE Web service functions *runscriptinVM* in the virtual machines.

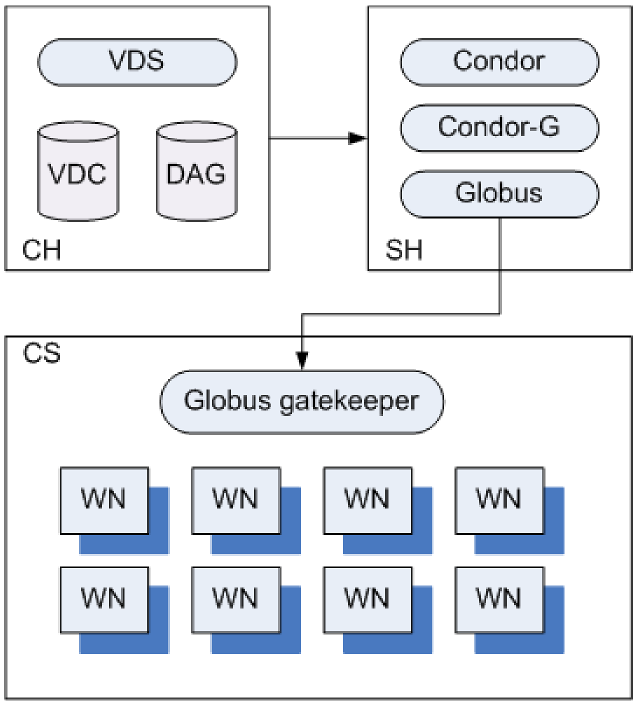


Figure 6.2: Grid model for VDS execution

6.3.2 Build a virtual cluster for Compute Site

To create a virtual cluster, configuration of a head node and several work nodes is required. On the head node, Globus Toolkit and Condor master are installed. On the work nodes, Condor workers are installed. Globus Toolkit *gatekeeper* in the head node is used to communicate with clients at Submit Hosts, for example, get jobs and return job results. Condor is used in the virtual cluster as a local batch scheduler to schedule jobs across multiple work nodes. The following processes are required to construct the virtual cluster:

Require a virtual machine

To configure a head node a virtual machine with Linux operating system is required. An XML file in GLUE schema is used to describe the virtual machine. For example, the XML file below is organized in GLUE schema and requires a virtual machine with SUSE10.1 operating system, 512M memory and AMD Opteron250 processor:

```
<VirtualMachine>
    <OperatingSystemName>SUSE</OperatingSystemName>
    <OperatingSystemRelease>10.1</OperatingSystemRelease>
    <PowerState>poweroff</PowerState>
    <MainMemoryRAMSize>512</MainMemoryRAMSize>
    <ProcessorVendor>AMD</ProcessorVendor>
    <ProcessorModel>Opteron250</ProcessorModel>
</VirtualMachine>
```

Users can invoke GVE Web service function *requestVM*, which takes the above GLUE XML file as a parameter and returns a virtual machine to users.

Globus Toolkit on the cluster head node

Now with a virtual machine on hand, users can install Globus Toolkit and Condor master service to build a cluster head node.

Before the installation, staging of required software packages is demanded, Globus Toolkit and Condor, from the FTP server to the virtual machine by calling Virtualization Service function *copyfiletoVM*.

The installation is processed automatically with the following example script with the Virtualization Service function *RunScriptInVM*:

```
useradd -d /home/globus -p globus -s /bin/bash  globus;
su - globus;
tar zxvf /tmp/gt4.0.1-all-source-installer.tar.gz;
export GLOBUS_LOCATION=$HOME/globus_location;
cd globus_installer;
configure;
make;
make install;
```

Some configuration work could not be automatically processed and human interaction is still required.

Build a Condor cluster

A condor cluster could be built automatically by running the following example script with the Virtualization Service function *RunScriptInVM*:

```
useradd -d /home/condor -p condor -s /bin/bash condor;
su - condor;
export CONDOR_HOME=/home/condor/condor_install_dir;
export CONDOR_CONFIG=
/home/condor/condor_install_dir/etc/condor_config;
export CONDOR_INSTALL_DIR=/home/condor-6.6.11;
tar zxvf condor-6.6.11-linux-x86-glibc23-dynamic.tar.gz;
$CONDOR_INSTALL_DIR/condor_configure \
--install=./release.tar --install-dir=$CONDOR_HOME <input.txt
```

6.3.3 Build a Submit Host

On the Submit Host, users require a Replica Location Service (RLS) [14, 161] to map abstract data files to concrete data files. A personal Condor, which includes the DAGMan and Condor-G, is used to manage workflows and submit workflows to virtual Grids. Globus Toolkit is also needed for job execution on Grids.

Build Globus Toolkit & personal Condor

The Submit Host needs a personal Condor with Condor-G to manage workflows and Globus Toolkit to submit workflows to computational Grids. Similar scripts for building Compute Site are re-used.

Build the Replica Location Service

Replica Location Service (RLS) is a simple registry which keeps records of where replicas exist on physical storage resources. The logical file name is the unique identifier for the contents of a data file. The physical file name is the location of a copy of the data file on a storage system. The function of RLS is to maintain mappings, or associations, between logical file names and one or more physical data file names of replicas. A user could provide a logical file name to the RLS server and request all the registered physical file names of replicas. The user could also query the RLS server to search the logical file name associated with a particular physical file location.

RLS is difficult to install and the configure as various required software packages are not compatible. After several tests, the following software package versions are employed as a stable distribute software bundle in the RLS installation:

- MySQL 4.0.27 as database back end of RLS server;
- MyODBC 3.51.06 as the ODBC driver for MySQL 4.0.27;
- iodbc 3.51.2 as the interface to the MyODBC of the MySQL

Example installation scripts are shown as follows:

```
get odbi.ini;
#install mysql 4.0.27
groupadd mysql;
useradd -p mysql -g mysql mysql;
cd /usr/local;
tar zxvf /tmp/mysql-standard\
-4.0.27-unknown-linux-gnu-x86_64-glibc23.tar.gz
ln -s\
mysql-standard-4.0.27-unknown-linux-gnu-x86_64-glibc23 mysql
cd /usr/local/mysql;
scripts/mysql_install_db --user=mysql;
chown -R root . ;
chown -R mysql data;
chgrp -R mysql .
#install MyODBC 3.51.06
su - globus;
tar zxvf /home/lizhe/download/MyODBC-3.51.06.tar.gz;
./configure --prefix=$GLOBUS_LOCATION \
--with-mysql-libs=$GLOBUS_MYSQL_PATH/lib \
--with-mysql-includes=$GLOBUS_MYSQL_PATH/include\
--with-iodbc=$GLOBUS_LOCATION\ --with-odbc-ini=$ODBCINIDIR;
make;
make install;
```

6.3.4 Start the Grid platform

Daemons of various software packages should be started to start Grid and cluster services, for example, Globus gatekeeper, Condor master, RLS service. Starting daemons could be finished by running scripts inside virtual machines for example:

```
#Compute Site
start globus gatekeepter;
start Condor master daemon;
#Submit Host
start globus gatekeepter;
start Condor master;
start MySQL database server;
start RLS server;
```

6.4 Build VDS on the virtual Grid infrastructure

6.4.1 Automatic installation of VDS package

The VDS packages are installed in the Submit Host with following script:

```
cd /home/vds;
ln -s vds-1.4.4 vds;
export VDS_HOME=/home/vds
source $VDS_HOME/setup-devel-env.sh;
ant clean dist;
```

6.4.2 VDS configuration with virtual Grid infrastructure

The following interfaces should be automatically configured, so that VDS could run on the virtual Grid infrastructure, as indicated in Section 6.3:

- Site Catalog
  Site Catalog defines the Compute Site profiles. As the Grid platforms are installed and configured with some pre-defined installation scripts, Site Catalog therefore could be automatically generated in the process of Grid construction. A sample Site Catalog in text format is shown as follows:

  ```
  pool VM1{
  lrc "rls://iwr-lizhe2.ka.fzk.de"
  workdir "/home/lizhe/workdir"
  gridftp
  "gsiftp://iwr-lizhe-vm1.ka.fzk.de/home/lizhe/storage" "4"
  universe transfer
  "iwr-lizhe-vm1.ka.fzk.de/jobmanager-fork" "4"
  universe vanilla
  "iwr-lizhe-vm1.ka.fzk.de/jobmanager-fork" "4"
  gridlaunch "/home/lizhe/vds/dist/vds-1.4.4/bin/kickstart"
  profile env "VDS_HOME" "/home/lizhe/vds/dist/vds-1.4.4"
  profile env "GLOBUS_LOCATION" "/home/globus/gt_4.0.2"
  profile env "LD_LIBRARY_PATH" "/home/globus/gt_4.0.2/lib"
  profile env "JAVA_HOME" "/usr/local/java"
  profile env "CLASSPATH"
  "/home/lizhe/vds/dist/vds-1.4.4/lib/gvds.jar:
  /home/lizhe/vds/dist/vds-1.4.4/lib/rls.jar:
  /home/lizhe/vds/dist/vds-1.4.4/lib/java-getopt-1.0.9.jar"
  }
  ```

Then the VDS tool *genpoolconfig* is executed to convert the Site Catalog from the text format to the XML format.

- Transform Catalog and Replica Catalog
  Transform Catalog and Replica Catalog are used to record application executables and data. They are configured when an application is organized in VDS.

6.5 GWVB: Grid Workflow in a Virtual Box

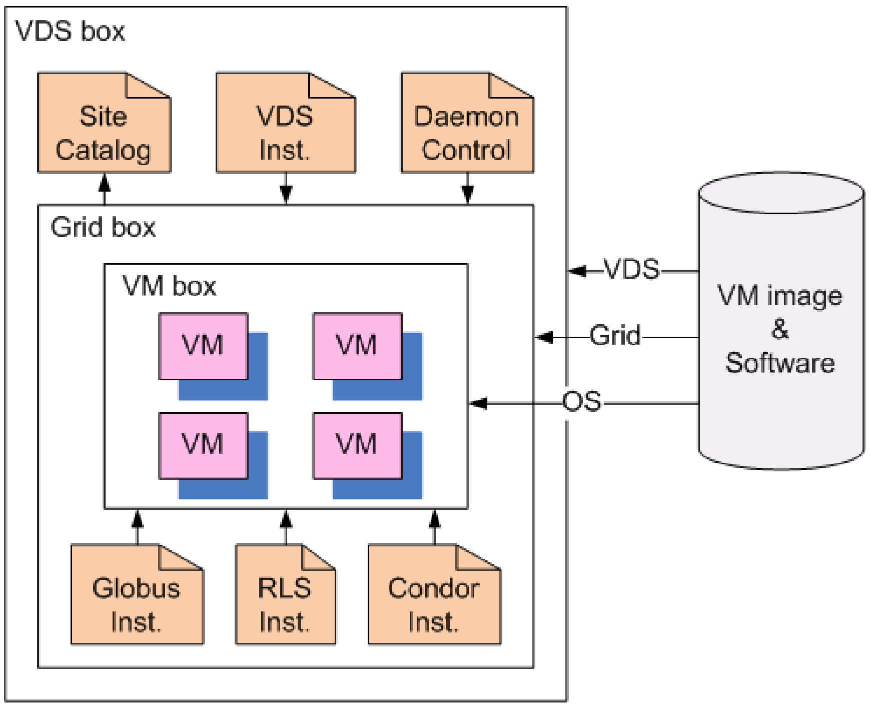


Figure 6.3: Grid Workflow in a Virtual Box

We define a Virtual Box as a software system, which contains data and the operations that work on the data. To start a process of the Virtual Box, users can trigger the Virtual Box with a signal or with several parameters.

Grid Workflow in a Virtual Box (GWVB) organizes installation, configuration and execution of a Grid workflow system as an automatic processes without interactive instructions from resource providers or system administrators.

Now we have some details in hand:

- pre-configured virtual machines,

- scripts for Globus & Condor installation, and
- scripts for VDS installation.

Conceptually, steps can be hierarchically grouped for VDS installation on virtual machines in several Virtual Boxes:

- Virtual Machine Box
  In this Virtual Box users run GVE Web service clients to create multiple virtual machines with desired operating systems, e.g., SUSE 10 in our case.
- Virtual Grid Box
  Grid middleware or distributed services could be installed on these virtual machines. In our scenario, a Grid computing system is built with Globus Toolkit as middleware. Globus Toolkit, personal Condor and RLS are installed on the client side. Globus Toolkit and Condor cluster are installed on the server side. Therefore with Virtual Machine Box and scripts for Grid middleware, installation builds Virtual Grid Box.
- VDS Box
  Application level software is then installed on the Virtual Grid box, e.g., here VDS. Users thus can build VDS workflows and execute them on the virtual Grid system. The VDS Box is composed of the Virtual Grid Box and the scripts for VDS installation .

6.6 Test case and discussion

6.6.1 Introduction to CMS computing software

The CMS (Compact Muon Solenoid) benchmark used here as a test case is CMS OO (namely OSCAR and ORCA) [117]:

- Monte-Carlo events are generated and stored in CMKIN [116],
- The CMS detector is described and its interaction with particles are simulated in OSCAR [100], and
- Simulated events are digitized in ORCA [115].

The CMKIN [116] software package provides a common interface between physics event generators and CMS detector simulation. It offers a standard method of interfacing kinematics generators, like PYTHIA, ISAJET and HERWIG, with CMS detector simulation. The interface is based on a common block HEPEVT – a HEP standard to store particle kinematics information for one event. CMKIN generates physics events for OSCAR detector simulation input.

OSCAR [100] simulates for CMS analysis and reconstruction. It reads events from CMKIN *Ntuples RawParticle*, uses GEANT-4 to simulate particle propagation through space and interaction with the detector material. OSCAR creates *SimHits* representing the information and finally stores *RawParticle*s, *SimTracks*, *SimVertices* and *SimHits* in POOL data files. The test uses OSCAR 3.7.0 , which simulates 300 single pion events of $50GeV/eP_t$.

ORCA [115] is a framework for reconstruction and is intended to be used for final detector optimizations, trigger studies or global detector performance evaluation. ORCA-digitization simulates the response of the readout electronics, which are the events simulated with OSCAR. The test employs ORCA 8.7.1, where events are digitized and reconstructed with the corresponding ORCA applications: *writeAllDigis* and *writeDST*.

To evaluate full performance of virtual machines, we also include several ROOT [117] based tests and the Monte Carlo generator Pythia [117] in the benchmark.

ROOT provides various benchmark routines:

- bench
  This test program compares the I/O performance obtained with all STL collections of objects or pointers to objects and also ROOT collection class *ClonesArray*.
- stressLinear
  The suite of programs tests many elements of the vector, matrix and matrix decomposition classes (matrix size 100×100).
- stressgeom
  Stressgeom tests the ROOT geometry classes.
- stress
  The suite of programs tests the essential parts of ROOT. In particular, there is an extensive test of the I/O system and Trees.

For the Pythia benchmark, we process 100000 super symmetry events at$\sqrt{s} = 14TeV$ (the Pythia *main65.f* for example). The code is compiled with *g77* with the *O2* option:

g77 -O2 -o main65 main65.f pythia6227.o

6.6.2 Generate CMS abstract workflow

The abstract workflow is organized as shown in Figure 6.4. CMKIN reads *datacard* from MC generator and generates events in *HEPEVT Ntuple*. OSCAR simulates HEP events and generates *pool data*: *Pool SimHits/minbias* and *Pool SimHits/signal*. ORCA digitizes and reconstructs the events that are simulated by OSCAR. ORCA outputs the *ROOT Tree*, which can be further employed for ROOT benchmark.

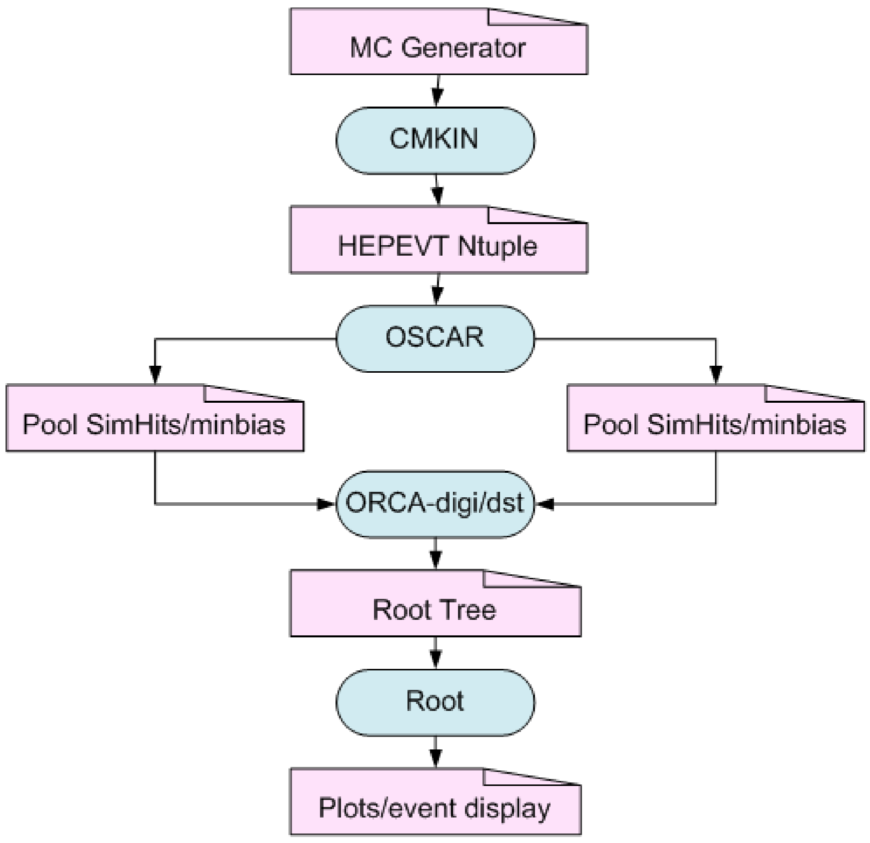


Figure 6.4: CMS workflow

6.6.3 Generate CMS concrete workflow

The following interfaces are developed to generate a concrete workflow:

- Transform Catalog
  Virtual machines containing CMS software are organized with packages inside. All software packages register themselves in the Transform Catalog file:

```
VM1  CMKIN      /cmstest/oscar/kine_make_ntupl.exe
VM1  OSCAR      /cmstest/oscar/script
VM1  ORCA-digi  /cmstest/orca/digi
VM1  ORCA-dst   /cmstest/orca/dst
```

- Replica Catalog
  The input/output data of software packages are registered in the RLS server, for example:

```
mccard  gsiftp://vm1.fzk.de/cmstest/CMKIN/mccard_zw_31.txt
zw_30   gsiftp://vm1.fzk.de/cmstest/CMKIN/zw_30.ntupl
```

- Site Catalog
  The Site Catalog was created when the virtual Grid infrastructure is constructed. This is described in Section 6.4.2.

6.6.4 Test organization

CMS benchmark is executed on virtual machines with 64-Bit processors, which support both 32-Bit operating system and 64-Bit operating system. CMS benchmark is a legacy application that demands various software libraries. Therefore, we are interested in migrating pre-compiled 32-Bit CMS benchmark to 32-Bit and 64-Bit operating systems. The test is organized in the following modes [179, 180]:

- Legacy mode
  The 64-Bit architecture is installed with 32-Bit operating system and 32-Bit application runs in the 32-Bit operating system.
- Compatibility mode
  The 64-Bit architecture is installed with 64-Bit operating system and 32-Bit application runs in the 64-Bit operating system.
- Full 64-Bit mode
  The 64-Bit architecture is installed with 64-Bit operating system. Legacy applications are recompiled with 64-Bit libraries and run on the 64-Bit systems.

In the test, we run CMS benchmarks in legacy mode, compatibility mode and full 64-bit mode to evaluate the performance of distributed virtual machines. Table 6.1 shows the available operational modes for CMS tasks in the test.

Table 6.1: Operational mode for CMS OO benchmark

CMS benchmark	**Legacy mode**	**Compatibility mode**	**Full 64-Bit mode**
OSCAR	Yes	Yes	No
ORCA	Yes	Yes	No
Root	Yes	Yes	Yes
Pythia	Yes	Yes	Yes

6.6.5 Test results

Table 6.2 and Table 6.3 show the performance of CMS workflow in terms of *Event/Second*. In Table 6.2 , only 1 copy of workflow runs on the virtual machines. In Table 6.3, 2 copies of CMS workflows run simultaneously. Average values of the test results are obtained from 2 copies. The reason of simultaneous running of 2 copies is to make full use of dual cores of AMD opteron processor 250. Each test is executed with 15 times.

Table 6.2: CMS OO benchmark on virtual AMD opteron processor 250

	Legacy mode	**Compatibility mode**
OSCAR	0.0880	0.0897
ORCA-Digi	0.135	0.119
ORCA-Dst	0.974	0.931

Table 6.3: CMS OO benchmark on virtual AMD ppteron processor 250 (2 copies)

	Legacy mode	**Compatibility mode**
OSCAR	0.0892	0.0911
ORCA-Digi	0.137	0.118
ORCA-Dst	0.984	0.934

Table 6.4 shows test results of ROOT benchmark. We get the test results from 1 copy ROOT benchmark and average values of 2 simultaneously running copies. The test results are in term of *ROOT marks*.

Table 6.4: ROOT stress benchmark on virtual AMD opteron processor 250

	Legacy mode	**Compatibility mode**	**Full 64-bit mode**
1 copy	945.0	864.2	1227.4
2 copies (av.)	959.8	830.1	1196.1

Table 6.5 shows the test results of Pythia benchmark, which processes 10000 SUSY Events. Test results are obtained from 1 copy of Pythia benchmark and average values of 2 simultaneously running copies. The test results are in terms of *Event/Second*.

Table 6.5: Pythia benchmark on virtual AMD opteron processor 250

	Legacy mode	**Compatibility mode**	**Full 64-bit mode**
1 copy	105.30	103.96	121.38
2 copies (av.)	105.76	113.23	121.28

6.6.6 Performance evaluation

Evaluation of performance of CMS benchmark on virtual machines is performed by comparing the test results on virtual machines (Table 6.2 - Table 6.5) with those on real machines. Test results on real machines are reported in CMS internal reports [179, 180]. Table 6.6 shows the results of running 1 copy of CMS benchmark on real AMD opteron processor 250.

Table 6.6: CMS benchmark on AMD opteron processor 250

CMS benchmark	Legacy mode	Compatibility mode	Full 64-bit mode
OSCAR	0.1186	0.1154	N/A
ORCA-digi	0.1562	0.1300	N/A
ORCA-DST	N/A	1.039	N/A
ROOT stress	957.6	898.3	1402.6
Pythia	121.3	120.8	146.9

Figure 6.5 – Figure 6.9 show the comparisons of OSCAR, ORCA-digi, ORCA-dst, ROOT and Pythia test results between virtual machines and real machines. In general, virtual machines can attain 70%-95% performance of real machines. ROOT benchmark achieves good performance on virtual machines, since ROOT stress is an I/O intensive application. Compared with ROOT benchmark, results from OSCAR tests are unsatisfactory as OSCAR is a computationally intensive application. Comparison on different modes is also tested. It could be concluded that legacy mode and compatibility mode achieve almost the same performance. Full 64-bit mode can achieve the best performance among the three modes. It is rather obvious because in the full 64-bit mode, 64-bit applications receive access to the full physical memory range and also allowed access to the new General Purpose Registers (GPRs) as well as the expanded GPRs in 64-bit processors.

6.6.7 Discussion

In addition the following questions are further considered:
Is the Grid Workflow in a Virtual Box (GWVB) really feasible?
Specifically, can users remotely submit a GWVB without too much network delay?

With regard to the example in this chapter, we can estimate it as follows:

- 8 virtual machine images are remotely submitted to form a Condor cluster;
- In each virtual machine, for example, we need 4 GB disk image for operating system;
- System software, e.g., Globus and Condor, and application level software, e.g., VDS, could be estimated less than 2 GB.

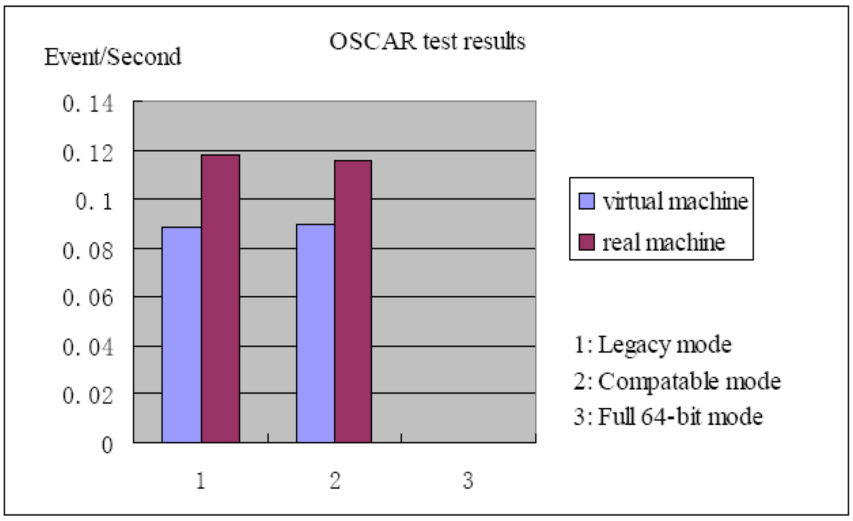


Figure 6.5: OSCAR test results

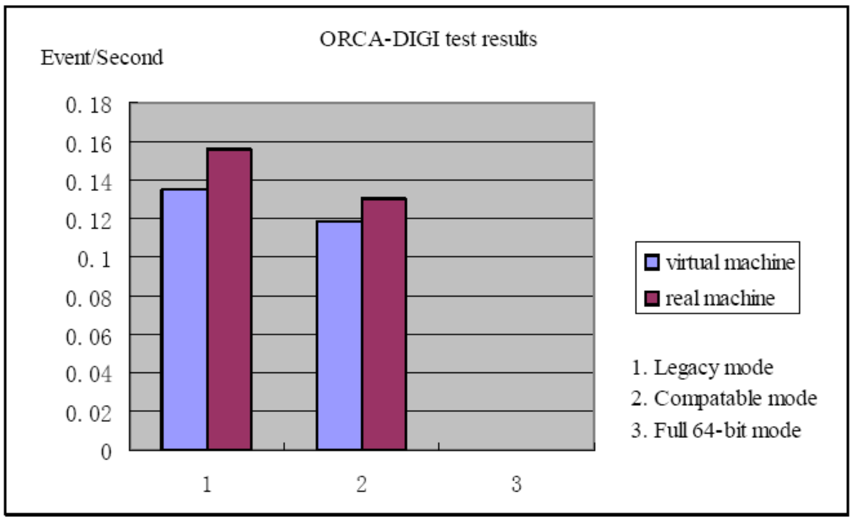


Figure 6.6: ORCA-digi test results

- CMS benchmarks with workflow are approximately 4 GB.

Two choices are available:

- submit 8 pre-configured virtual machine images (around 80 GB) to remote computer site, or

- submit 1 pre-configured virtual machines (around 10 GB) to remote computer site, then make 7 duplicates on remote site.

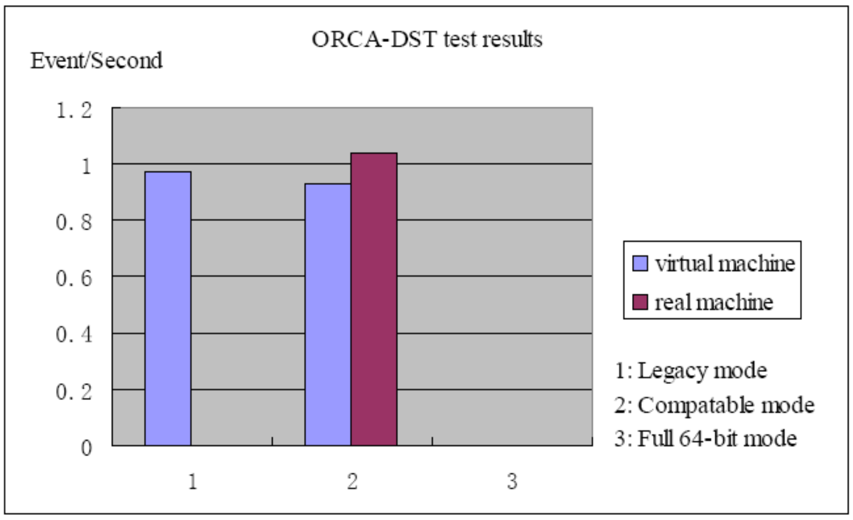


Figure 6.7: ORCA-dst test results

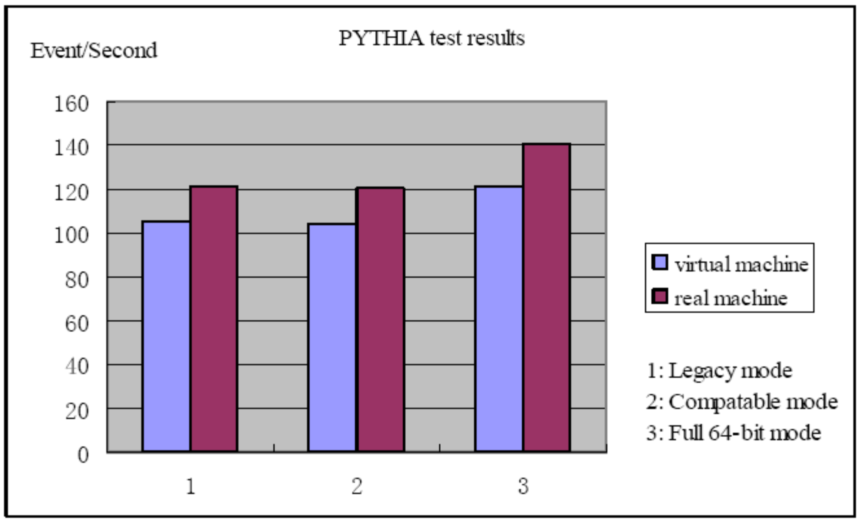


Figure 6.8: Pythia test results

If the network bandwidth between computing centers is 1 MB/s, then transfer of 1 virtual machine image of GWVB is around 10000 seconds (3 hours). If the network bandwidth between computing centers is 10 MB/s, then transfer of 1 virtual machine images of GWVB is around 1000 seconds (18 minutes). Considering the facts that available communication network bandwidth between compute centers are more than 10 MB/s, transfer of a GWVB is very likely and feasible.

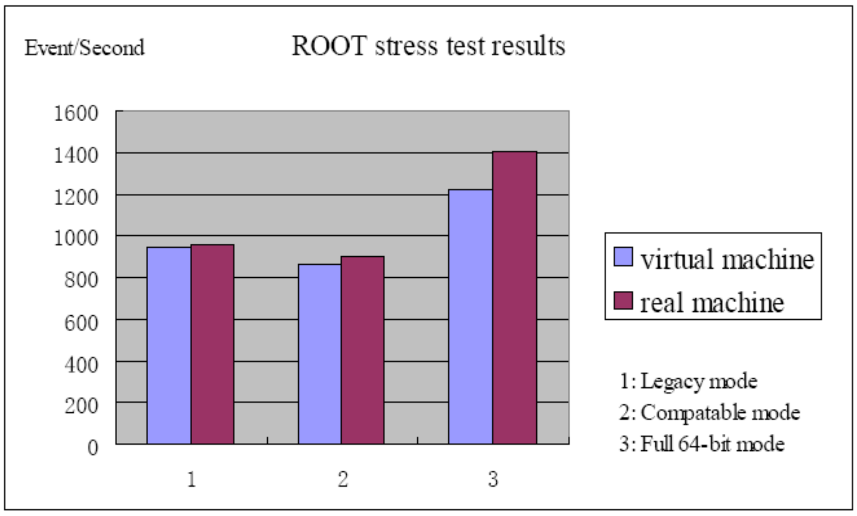


Figure 6.9: ROOT test results

6.7 Summary

This chapter presents an example of VDE philosophy – to construct a Grid Workflow system in a Virtual Box (GWVB). The Grid workflow in our example is composed of:

- underlying virtual machines as Grid infrastructures,
- various middleware packages, e.g., Globus and Condor,
- Grid workflow engine, VDS in our context, and
- Grid workflow application, CMS, for example.

Grid workflow system therefore could be put into several hierarchical Virtual Boxes depending user requirements:

- Virtual machine Box,
- Virtual Grid Box, and
- Grid workflow system in a Virtual Box with the workflow applications.

A GWVB could be remotely submitted, executed and migrated to a computing center. A GWVB is easy to steer as it contains both virtual machine images of Grid platforms and operation scripts that configure and deploy the Grid workflow system. The new methodology thus renders great facilities to Grid users.

Chapter 7

Virtual e-Science Infrastructure on Demand: a Novel Methodology

7.1 Introduction

This chapter presents a procedure to dynamically organize an e-Science infrastructure on distributed virtual machines. An e-Science infrastructure contains GridSAM [118] as Grid middleware and ActiveBPEL engine [151] as workflow engine to build a workflow system. The Virtualization Service developed in Chapter 5 is utilized to provide virtual machines as computing resources for the e-Science infrastructure. This chapter emphasizes that a VDE could be dynamically built and deployed by merging Grid Virtualization Service into the middleware level and the application level. This use scenario also demonstrates that a VDE could employ any desired middleware, namely GridSAM here, which could differ from the middleware of underlying Grid infrastructures.

7.2 The e-Science infrastructure: an overview

7.2.1 GridSAM: job submission and monitoring

GridSAM [118] is a standard job submission and monitoring Web service, which provides a common interface to a variety of DRMs (Distributed Resource Managements). GridSAM is developed with widely accepted and standardized Web Service specifications and related technologies. GridSAM uses the Job Submission Description Language (JSDL) specification from the Open Grid Forum [153].

The objective of GridSAM is to allow users to execute applications on existing distributed resource managers transparently and monitor their status. Transparency is achieved through the use of a common job description language, JSDL and a uniform networked access interface, WS-I compliant Web Service. The core function of GridSAM is to translate the submission instructions into a set of resource-specific actions: file staging, launching and monitoring. It aims to minimize user attention in the job launching process with a minimal overhead. The Web Service interface allows jobs to be submitted as described in a JSDL document and status retrieved as a chronological list of events – the state of the job.

GridSAM Web service offers the following *port*s in the WSDL file:

- *JobSubmissionPortType* provides the job submission functions,
- *JobMonitoringPortType* defines the operations for retrieving job status, and
- *JobControlPortType* supports functions such as job start and job stop.

7.2.2 ActiveBPEL workflow engine

ActiveBPEL engine [151] is a modeling, monitoring and execution environment for scientific workflows based on the Business Process Execution Language (BPEL) [149]. BPEL is the de-facto standard for the composition and orchestration of Web services. There are a number of advantages from adaption of BPEL for the orchestration of scientific workflows. The standard is implemented in several industrial strength competing products by IBM, Oracle, ActiveEndpoints and others. Due to BPEL's focus on business rather than scientific processes, implementations of the BPEL standards have not yet been tested for large and long running scientific workflows in a Grid environment.

The OMII-BPEL project [151] has developed a production-strength modeling, monitoring and an enactment environment for scientific workflows based upon the BPEL. The environment incorporates a version of the ActiveBPEL engine that is customized and tested by a number of UK e-Science projects for scientific workflows. To model workflows the environment contributes specific extensions to the BPEL Designer open source tool. The BPEL Editor inherits many features from the Eclipse platform, such as integration with CVS, platform specific look and feel, repository integration and the Web tools platform, which provides support for WSDL and XML Schema editing.

7.2.3 e-Science infrastructure with GridSAM and ActiveBPEL

A typical e-Science infrastructure [151] that involves BPEL (both the BPEL script and BPEL runtime) and GridSAM is included in the following (see also Figure 7.1):

- use BPEL Designer to design a BPEL process that interacts with GridSAM's job submission service *port* and job monitoring service *port*; produce the deployment archive by BPEL Designer at the end of modeling;

- deploy the process on ActiveBPEL, which is hosted in OMII Server container; construct the request message from the BPEL Designer that triggers the BPEL process;
- once started, ActiveBPEL engine submits a pre-defined job in JSDL to GridSAM; GridSAM translates the JSDL script to whatever is compatible with the related resource manager and sends the job to the underlying Grid resources;
- ActiveBPEL polls the job status through GridSAM's monitoring interface until the job is completed.

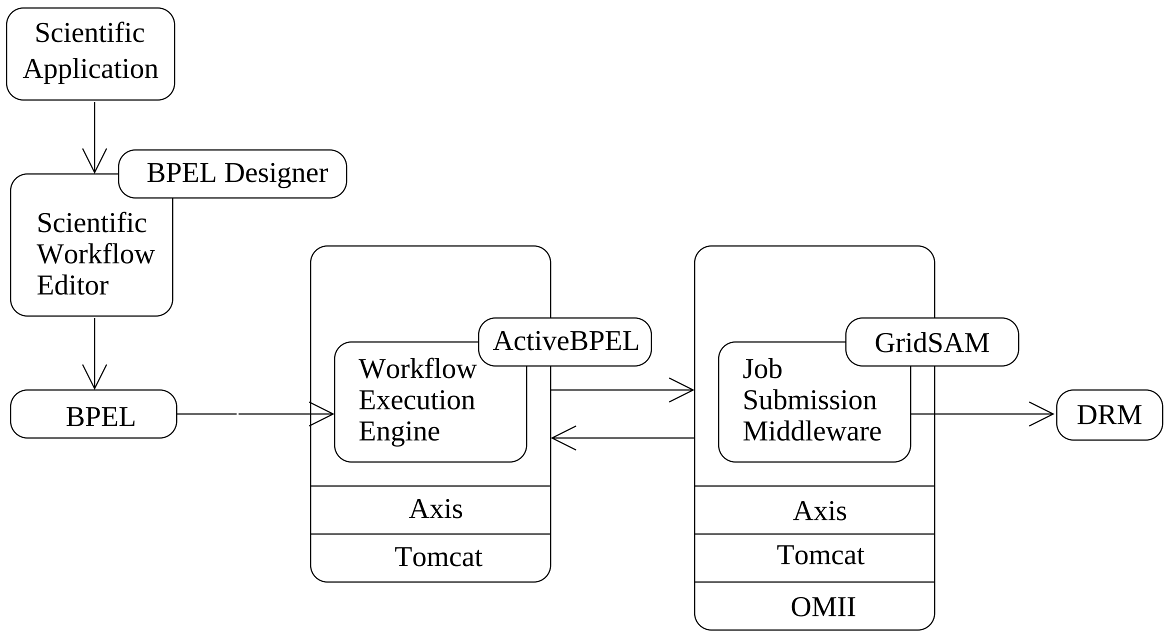


Figure 7.1: A typical e-Science architecture with GridSAM and ActiveBPEL engine

7.3 Build a virtual e-Science infrastructure at runtime

7.3.1 System integration

Virtual machines are employed as computing resources for BPEL workflow execution. ActiveBPEL engine dynamically invokes Grid Virtualization Service to request virtual machines with GridSAM installation, then organizes the application in workflow and submit the workflow to virtual machines via GridSAM Web service. The integrated workflow system includes the following components:

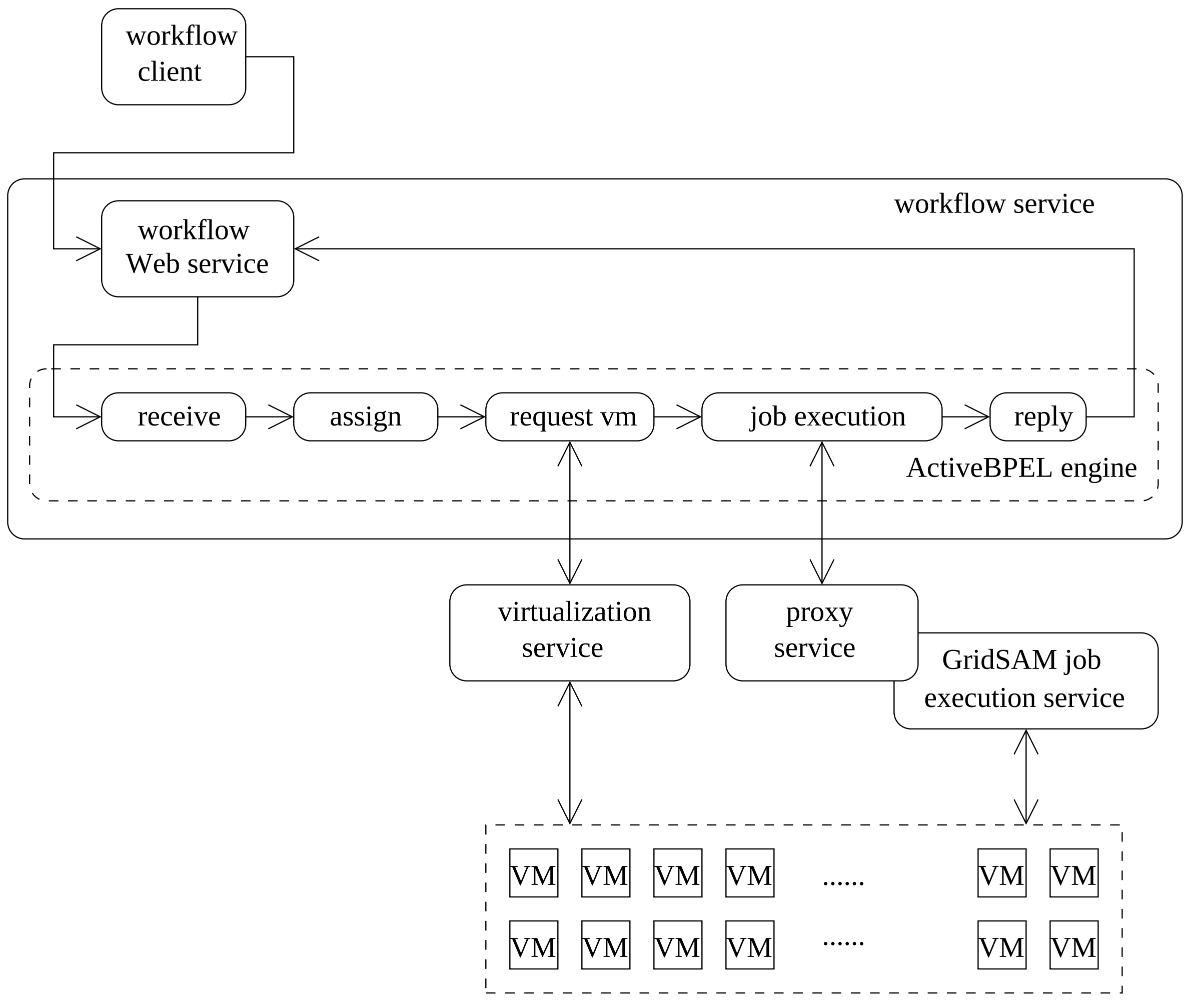


Figure 7.2: Service composition of virtual e-Science Infrastructure

- Workflow Web service
  The workflow service contains a Web service as interface, which could be invoked by a workflow client. The ActiveBPEL acts as a workflow engine. It invokes the Grid Virtualization Service to request virtual machines with GridSAM installation, then executes workflow jobs on virtual machines via the GridSAM interface.

- Proxy service
  A proxy service is implemented as the interface between the ActiveBPEL engine and the GridSAM service.

- GridSAM service
  GridSAM job submission and monitoring services are installed in the virtual machines, through which the ActiveBPEL engine submits jobs to virtual machines.

7.3.2 How does ActiveBPEL invoke Virtualization Service?

The ActiveBPEL engine invokes the Grid Virtualization Service to obtain virtual machines. In order to invoke a Web service (e.g., the Virtualization Service here), the ActiveBPEL engine complete the following steps:

- define *partnerLinkType*
  The Virtualization Service should firstly be declared in the BPEL workflow by defining a *partnerLinkType* in the workflow Web service description file. The declaration of *partnerLinkType* describes the Virtualization Service to be invoked, e.g., *name*, *namespace*, and *portType*. For example, Grid Virtualization Service is declared as follows:

  ```
  <plnk:partnerLinkType
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  name="GVEngineLT">
  <plnk:role name="GVEngineLTRole" portType="ns1:GVEngine"/>
  </plnk:partnerLinkType>
  ```

- define *PartnerLink*
  *PartnerLink*s describe the roles that a process or a Web service fulfill and the data it manipulates. With the *PartnerLinkType* defined above, the *PartnerLink* for the Virtualization Service is shown as follows:

  ```
  <bpel:partnerLinks>
      <bpel:partnerLink
          myRole="GVEngineLTRole"
          name="GVEngineLT"
          partnerLinkType="ns1:GVEngineLT"
          partnerRole="GVEngineLTRole"/>
  </bpel:partnerLinks>
  ```

  The example shown above gives details about the location of the WSDL document of the Virtualization Service Web service.

- Invoke the Virtualization Service operations
  The ActiveBPEL engine can therefore parse the document in order to discover all the operation provided by the Virtualization Web service with *partnerLinkType* and *PartnerLink* declarations. The following example indicates the procedure of invoking the *requestVirtualMachine* operation of the Virtualization Service. The input and output variables, *requestVirtualMachine* and *requestVirtualMachineResponse*, are declared in the part $< bpel : variable >$ of the BPEL document. The operation of *requestVirtualMachine* is invoked by using a $< bpel : invoke >$ activity; the *partnerLink* attribute indicates which Web service is addressed. The *portType* attribute specifies the *portType*

that contains the operation invoked. The operation attribute contains the name of the invoked operation. In this example, the *requestVirtualMachine* operation is invoked.

```
<bpel:variables>
    <bpel:variable
        messageType="requestVirtualMachine"
        name="requestVirtualMachine"/>
    <bpel:variable
        messageType="requestVirtualMachineResponse"
        name="requestVirtualMachineResponse1"/>
</bpel:variables>
<bpel:invoke
    inputVariable="requestVirtualMachine"
    name="Request_VM1"
    operation="requestVirtualMachine"
    outputVariable="requestVirtualMachineResponse1"
    partnerLink="GVEngineLT"
    portType="GVEngine"/>
```

7.3.3 GridSAM proxy service

The ActiveBPEL engine cannot dynamically invoke GridSAM Web service in that:

- As virtual machines are requested dynamically, GridSAM Web services that are installed on virtual machines could only be identified at runtime. When the ActiveBPEL engine organizes a workflow, the Virtualization Service has not yet been invoked by the ActiveBPEL engine. The GridSAM Web services could be installed on the virtual machines only after the Virtualization Service is invoked. Therefore, the endpoints of the GridSAM Web services could not be available when the ActiveBPEL engine composes the workflow.
- The GridSAM Web service is secured using WS-Security, which is unfortunately not supported by the current release of ActiveBPEL engine.

A proxy service is developed in the e-Science infrastructure to solve the above issues as follows:

- The access to GridSAM proxy service is not secured using WS-security, therefore allows non-security interaction with the ActiveBPEL engine.
- The proxy service takes the GridSAM service endpoints of the jobs to be executed as input parameters. Its function is to invoke the GridSAM Web service indicated by the input endpoint, by submitting the job which is described in the input parameters.

The ActiveBPEL engine does not directly invoke the GridSAM Web service. The ActiveBPEL engine invokes Grid Virtualization service, then at runtime receive a set of endpoints of GridSAM Web service installed on the virtual machines. It then invokes the proxy service and transfers the endpoint of GridSAM Web services to be invoked as parameters to the proxy service. The proxy service thereafter invokes the GridSAM Web services instead.

The proxy service implements the following functions:

- Job submission
  To submit a job to a GridSAM service, the workflow calls the WSDL operation *submitJob*, which is shown below:

```
<xs:complexType name="submitJob">
    <xs:sequence>
<xs:element name="GridSAM_ENPOINT_ADDRESS" type="string"/>
<xs:element name="jobDefinition"
            type="JobDefinition_Type"/>
    </xs:sequence>
</xs:complexType>
```

  The first element of the input parameter is the GridSAM endpoint address. It identifies the GridSAM service where a job is submitted. The second parameter is the job definition. It is the description of the job to be executed. The *submitJob* operation returns a WSDL message describing the state of the job:

```
<xs:complexType name="gridSAMProxyJobStateType">
    <xs:sequence>
<xs:element name="ID" type="string"/>
<xs:element name="state"
    type="gridSAMProxyJobStatusEnum"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="gridSAMProxyJobStatusEnum">
    <xs:restriction base="xs:string">
        <xs:enumeration value="SUCCESS"/>
        <xs:enumeration value="ERROR"/>
        <xs:enumeration value="RUNNING"/>
        <xs:enumeration value="QUEUED"/>
    </xs:restriction>
</xs:simpleType>
```

The first element of the returned message is the executed job ID. It will be used by the ActiveBPEL engine to monitor the job. This identifier is generated by the proxy service and is unique. The second element of the returned message is the current state of the job, *gridSAMProxyJobStatusEnum*.

- Job monitoring definition
  Job monitoring is achieved by calling the WSDL operation *waitForJobToTerminate*. The input parameter is the job ID monitored. The output of this operation is the current state of the job.

The proxy service acts also as the client of GridSAM Web service, because it submits and monitors jobs on GridSAM Web service. The proxy service implements following classes to access a GridSAM service:

- The *ClientSideJobManager* class provides methods of submitting jobs to a given GridSAM Web service and monitors the jobs.
- The *JobInstance* class provides the state of a running job.
- The *JobDefinitionDocument* is an internal data structure that defines the job information.

To submit a job to a GridSAM Web service the proxy service executes the following steps:

- The proxy service instantiates a *ClientSideJobManager* object, whose parameters are the endpoint addresses of the GridSAM Web services, as well as the definitions of the jobs to be executed.
- The job is submitted to the GridSAM Web service by calling the *submitJob* method, which returns a *JobInstance*.
- The proxy service creates a job state, that will be returned to the user, which is equivalent to the job state returned by the GridSAM Web service.

The job monitoring operation of the proxy service, *waitForJobToTerminate*, is a synchronous operation. When the operation *waitForJobToTerminate* is called, the proxy service creates a *ClientSideJobManager* object to monitor the job. The *ClientSideJobManager* object queries the job status by calling *getJobStatus* operation of GridSAM service repeatedly.

7.4 Distributed bio-sequence analysis: a test case

7.4.1 Distributed bio-sequence analysis

Bio-sequence alignment is a common and often repeated task in molecular biology. The analysis process consists of finding similarities between a particular query sequence and all the sequences of a bio-bank. This operation allows biologists to point out sequences sharing common subsequences. From a biological point of view, it leads to the identification of similar functionalities. The need for accelerating this application comes from the exponential growth of the bio-sequence database: every year the size scales by a factor 1.5 to 2 [176].

To compare two sequences, it is demanded an algorithm that gives a measure of similarity between them. The algorithm needs to align both sequences, for example, insert spaces in arbitrary locations of one sequence, thus leading an optimal score. The score can be simply calculated by assigning a value when symbols at the same location match, a different value when they are different and a third one when a space is introduced in one of the sequences. Here is an example of bio-sequence alignment. Let consider two sequences `P = GATATGTAT` and `Q = GGATGATT`. It is assumed a score system that assigns 1 to match, -1 to mismatch and -2 for a space insertion or a gap. A possible alignment between the sequences is shown in as follows.

P:	G	A	T	A	T	G	T	A	T	□
Q:	□	G	G	A	T	G	□	A	T	T
score:	-2	-1	-1	1	1	1	-2	1	1	-2

The process of bio-sequence alignment executes the Smith-Waterman algorithm [114]. Let consider two strings P and Q of length m and n. To identify common subsequences, the Smith-Waterman algorithm computes the similarity $H(i,j)$ of two sequences ending at position i and j of the two sequences P and Q. The computation of $H(i,j)$ is given by the following recurrences:

$$H(i,j) = \max \begin{cases} 0 \\ E(i,j) \\ F(i,j) \\ H(i-1,j-1) + S_{bt}(P_i, Q_j) \end{cases} \tag{7.4.1}$$

$$E(i,j) = \max \begin{cases} H(i,j-1) - \alpha \\ E(i,j-1) - \beta \end{cases} \tag{7.4.2}$$

$$F(i,j) = \max \begin{cases} H(i-1,j) - \alpha \\ F(i-1,j) - \beta \end{cases} \tag{7.4.3}$$

where S_{bt} is a character substitution cost table,

$$1 \leq i \leq m$$

$$1 \leq j \leq n$$

Initialization of these values are given by:

$$H(i,0) = E(i,0) = 0$$

$$H(0,j) = F(0,j) = 0$$

Multiple gap costs are taken into account as follows: α is the cost of the first gap; β is the cost of gaps. Let assume that:

$$\alpha = 1$$

$$\beta = 1$$

$$S_{bt}(x,y) = \begin{cases} 2 & if(x = y) \\ -1 & if(x \neq y) \end{cases}$$

Following example illustrates how Smith-Waterman algorithm works. From the highest score (10 in the example), a trace back procedure delivers the corresponding alignment, the two subsequences `TCGTATGA` and `TCTATCA`.

	□	A	T	C	T	C	G	T	A	T	G	A	T	G
□	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	2	1	0	0	2	1	0	2
T	0	0	2	1	2	1	1	4	3	2	1	1	3	2
C	0	0	1	4	3	4	3	3	3	2	1	0	2	2
T	0	0	2	3	6	5	4	5	4	5	4	3	2	1
A	0	2	2	2	5	5	4	4	7	6	5	6	5	4
T	0	1	4	3	4	4	4	6	5	9	8	7	8	7
C	0	0	3	6	5	6	5	5	5	8	8	7	7	7
A	0	2	2	5	5	5	5	4	7	7	7	10	9	8
C	0	1	1	4	4	7	6	5	6	6	6	9	9	8

This task thus in nature is a data driven and computationally intensive application. To increase the processing speed, the query sequence is generally divided into multiple sub-sequences and distributed to numerous computing resources for processing (see also Fig. 7.3).

7.4.2 Organize the workflow

There are four Web services involved in the scientific workflow:

- the ActiveBPEL engine,
- the Grid Virtualization Service,
- the proxy service, and
- the GridSAM Web service.

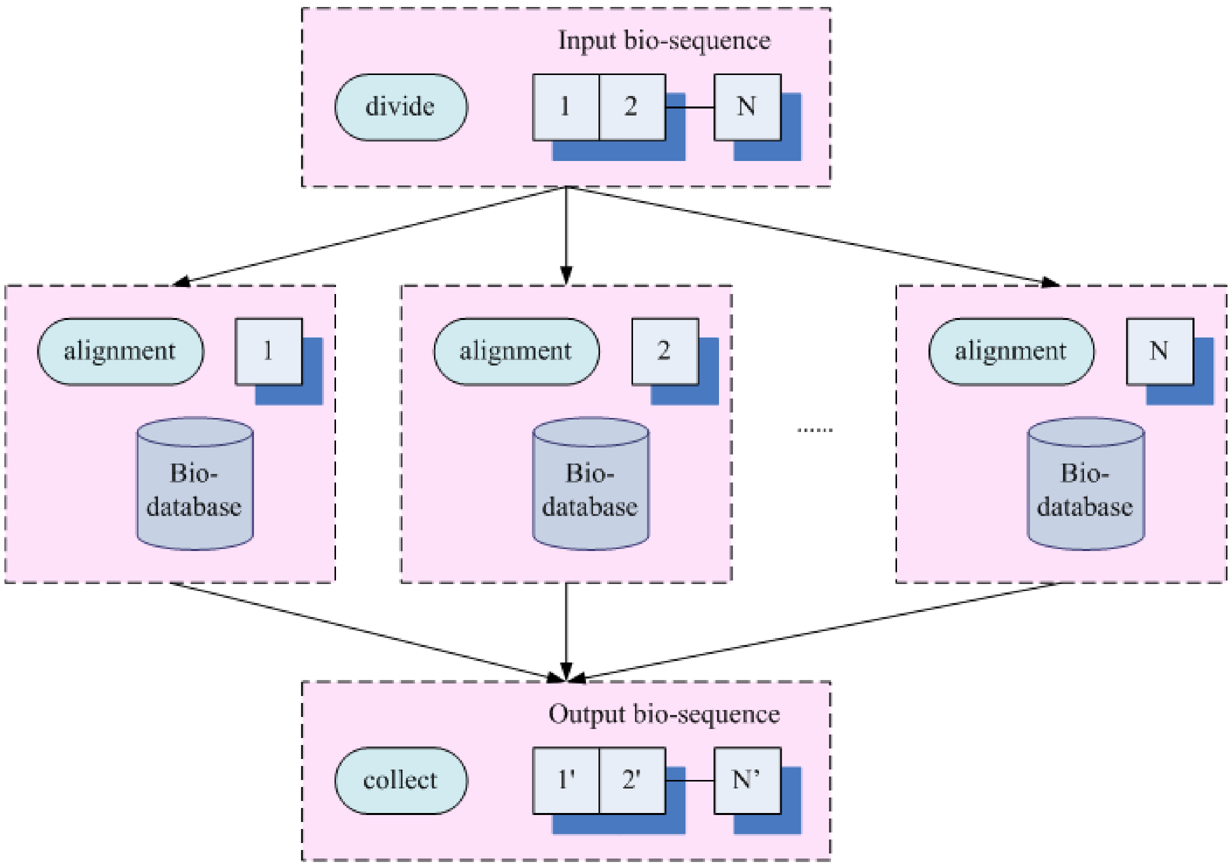


Figure 7.3: Workflow of distributed bio-sequence alignment

The execution process of these services is shown in Figure 7.4:

1. The workflow client submits a workflow to the ActiveBPEL workflow engine;
2. The ActiveBPEL workflow engine requires virtual machines from the Virtualization Service;
3. The Virtualization Service returns virtual machine profiles with GridSAM installation;
4. The ActiveBPEL engine resolves the GridSAM endpoints from the returned virtual machine profiles;
5. The ActiveBPEL engine submits workflow jobs to the proxy service with GridSAM endpoints as job parameters;
6. The proxy service submits jobs to GridSAM services;
7. The GridSAM service returns job status to the proxy service;

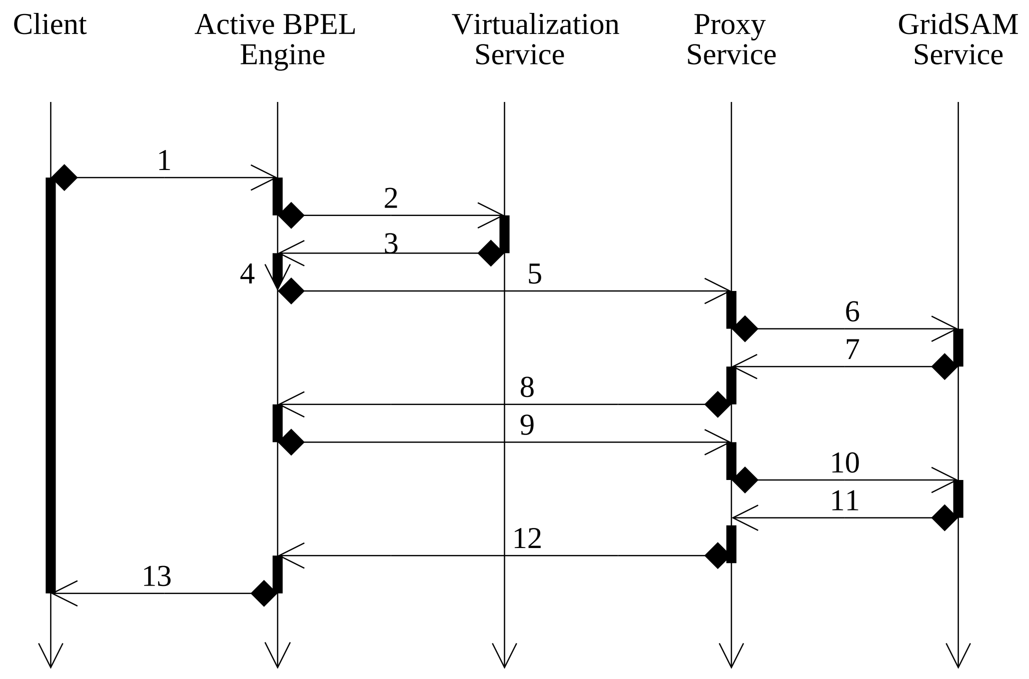


Figure 7.4: Execution process of the workflow services

8. The proxy service returns job status to the ActiveBPEL engine service;
9. The ActiveBPEL engine service calls the *WaitToJobComplete* operation of the proxy service to monitor the job;
10. The proxy service calls the *getJobStatus* operation of GridSAM service to retrieve the job status;
11. GridSAM service returns job status to the proxy service;
12. The proxy service returns job status to the ActiveBPEL engine service;
13. The ActiveBPEL engine service returns results to the client, when the workflow execution finishes.

GridSAM service is accessed via the proxy service, three *PartnerLink*s are defined in the workflow:

- *GridSAMProxyLT* represents the proxy service,
- *GVEngineLT* represents the Grid Virtualization Service.
- *ProteinLT* that is the *PartnerLink*, represents the ActiveBPEL engine service.

The workflow is triggered when the user calls the *computeProteinReceive* operation that takes all job submission description XML documents as parameters (*install.jsdl*, *divide.jsdl*, *alignment.jsdl*, *collect.jsdl*). After internal parameter initialization, virtual machines are requested in parallel in a BPEL flow. The requested virtual machines are installed with software in the next BPEL flow. Installation process is carried out in parallel on two virtual machines. The bio-sequence alignment is computed concurrently in virtual machines.

7.4.3 Build the workflow client

The workflow client is responsible for starting the workflow by submitting all the required files to the ActiveBPEL engine. A BPEL workflow is a Web service composition which exposes itself as a Web service. Therefore any workflow user only needs to create a Web service client that communicates with the workflow Web service interfaces.

The workflow client is developed using JAX-WS (Java API for XML – Web Services) technology. The client is implemented by performing the following steps:

1. generate the Java Web service client from the WSDL file of the workflow Web service.

2. import the generated classes and call the Web service operations.

In following code skeletons, the *ProteinService* is the starting point to access the Web service. It has a method *computeProtein*, which invokes the operation of workflow Web service.

```
 // create an instance of the client interface Protein
protein = (new ProteinService()).getProteinServicePort();
// instantiate the workflow request parameter
ComputeProteinRequest payload = new ComputeProteinRequest();
// Set parameters for the payload
......
// start the workflow by calling the computeProtein
//and wait for the reply
ComputeProteinResponse
    response = protein.computeProtein(payload);
```

7.4.4 Test results

In the test, two virtual machines are dynamically invoked, **VM1** and **VM2**. The protein sequence alignment is processed on the virtual machines. The test results of the workflow execution are shown in Table 7.1. The *alignment* task is computationally intensive and demands many computing resources.

Table 7.1: Test results of distributed bio-sequence analysis

Execution module	**Computing resource**	**Turn-around time** (second)
divide	LZ02	< 60
alignment	VM1	28140
alignment	VM2	28143
collect	LZ02	< 60
the workflow	the test bed	28144

7.4.5 Discussion

The bio-sequence alignment application demonstrates interesting benefits of virtual e-Science infrastructure:

- Dynamic resource allocation
  As the virtual resources are dynamically required by e-Science applications, the resource providers are not responsible for resource allocation. The resource consumed by the bio-sequence alignment is proportioned to the input bio-sequence. Therefore the required resource quota could be dynamically determined before running the application.

- System automation
  In the traditional Grid computing paradigm, Grid resources should be allocated and reserved before running applications. Therefore the Grid computing paradigm contains several steps: resource requirement, computing environment configuration and job submission. In the paradigm of virtual e-Science infrastructure, above steps could packaged into a virtual infrastructure and executed automatically.

7.5 Summary

This chapter offers an interesting VDE use case – building a virtual e-Science infrastructure at runtime. In this VDE use scenario, the Virtualization Service is invoked dynamically to construct distributed virtual machines, then the user level middleware, GridSAM and OMII-BPEL, are installed and configured at runtime. The virtual e-Science infrastructure use case shows that a VDE could be built dynamically on demand by invoking the Virtualization Service.

Chapter 8

Virtual Computing Center: a Novel Computing Service Provision

8.1 Overview

This chapter discusses an interesting use case of VDE: operate on a virtual computing center. In this case, the remote computing center provides a great deal of virtual machines. A group of users could lease a set of virtual machines from the computing center and install some middleware on the virtual machines to build a virtual computing center. UNICORE is used in this example, which was introduced in Section 2.1.2, to manage the virtual computing center (see also Figure 8.1).

A virtual computing center functions in the same way as a real computing center. It provides portals or Web services from which users can access computing resources in the virtual computing center. The builder of the virtual computing center normally operates as the administrator of the virtual computing center. Application level software and libraries are provided on the computing resources inside the virtual computing center and defines the management policies and access control rules.

The use scenario offers some advantages to both computing center and users in following aspects:

- The computing center is released from the difficult tasks of resource management and software provision. Computing centers only support virtual virtual machines to users in terms of number and lease. This minimizes the daily tasks of resource management of a computing center. Computing centers are also free of management issues such as software copyright provision, since it does not offer application level software computing environment.

- Users can customize their computing environment for some specific application domains.

The administrator can define the computing environment with specific communication protocols, scheduling policies and security control. In the virtual computing center, users are also free to install some system level and user level software & libraries. The virtual computing center contains the same user interfaces as a real computing center.

- After the lease is expired, the virtual computing center could be stored as a bundle of virtual machine disk images and returned to the user. Users can continue their virtual computing center by shipping the virtual machine images to another computing center. Users obtain more abilities to control jobs and resources.

In this chapter, a sample virtual computing center instantiates that users can define their own security management policy, X.509 [174] certificate offered by OpenSSL [98], which is independent with underlying Grid security implementation.

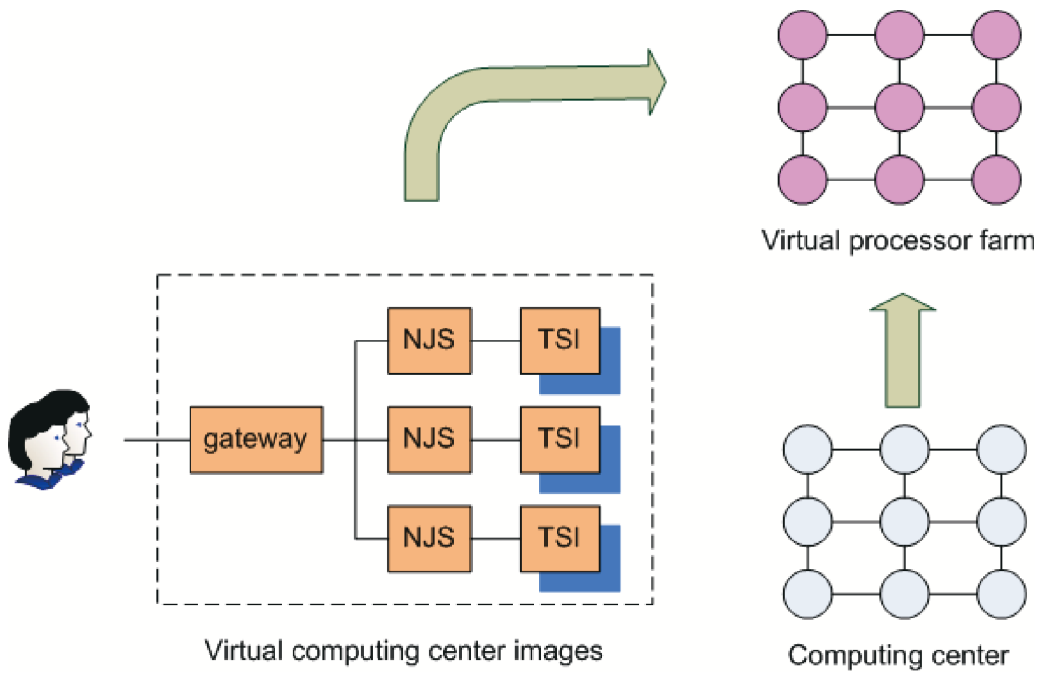


Figure 8.1: Virtual computer center

8.2 Manage a virtual computing center with UNICORE

8.2.1 Preparation of VDE disk images for the virtual computing center

The UNICORE has been selected as the middleware to access the virtual computing center. A virtual computing center is constructed with one UNICORE *gateway* and multiple UNICORE

*Vsite*s (see also Figure 8.1). Two types of virtual machine images should be prepared: UNICORE *gateway* and UNICORE *Vsite*.

UNICORE *gateway*

The UNICORE *gateway* is the access point of a virtual computing center for all UNICORE connections. It accepts data from clients and transfers it onto the UNICORE servers inside the site.

Prepare a *gateway* virtual machine disk image with the following steps:

- require a virtual machine with SUSE 10 Linux operating system and Java 1.4 pre-installation by invoking the *requestVM* operation.
- copy UNICORE *gateway* package, by invoking *copyfile* operation of Virtualization Service on the UNICORE *gateway* virtual machine. The UNICORE *gateway* package contains:
  - UNICORE *gateway* pre-installation packages;
  - SSL configuration files: *keystore* with private key and public certificates;
  - a *gateway* configuration file: *gateway.properties*.
    gateway.properties is pre-configured to allow dynamic registration of *NJS* and accept above *keystore* and public certificates. In the configuration file, the *gateway* contact information is left blank, which will be defined after the virtual computing center is set up.

UNICORE *Vsite*

A UNICORE *Vsite* is a virtual Compute Site which includes one computer server in our implementation. The *Vsite* virtual machine image is constructed as follows:

- require a virtual machine image with SUSE 10 Linux operating system and Java 1.4 & Perl 5.0 pre-installation;
- add into the virtual machine disk image a local account, "*unicore-worker*";
- copy *NJS* packages to the *Vsite* virtual machine, which contains:
  - UNICORE *NJS* pre-installation packages;
  - SSL configuration files: *keystore* with private key and public certificate;

  - a *NJS* configuration file: *njs.properties*.
    njs.properties is pre-configured to set *NJS* communication port and accept above *keystore* and public certificates. In the configuration file, the *gateway* contact information is left blank, which will be defined after the virtual computing center is set up.

- copy *UUDB* packages to the *Vsite* virtual machine and map the user public certificate to local account "*unicore-worker*";
- copy *TSI* packages and *TSI* configuration file, *tsi.properties*, to the *Vsite* virtual machine.

8.2.2 Map disk images to virtual processors

Disk images of a virtual computing center consist of one UNICORE *gateway* virtual machine image and multiple identical UNICORE *Vsite* virtual machine images. These images are started up by virtual machine hypervisors in the computing center and thus get the IP address of virtual machines. Two post-configurations should be performed:

- *gateway* configuration: put the IP address and port number of *gateway* virtual machine in the *gateway.properties*.
- *NJS* configuration: put the IP address and port number of *gateway* virtual machine in the *njs.properties*.

To activate the virtual computing center, daemons of *gateway*, *NJS* and *TSI* need to be started by invoking *RunScriptInVM* operations of GVE. Following scripts inside virtual machines need to be executed:

- *start_gateway.sh* in UNICORE *gateway* virtual machine,
- *start_njs.sh* and *start_tsi.sh* in UNICORE *Vsite* virtual machines.

8.3 Access control customization

8.3.1 Introduction

The virtual computing center is accessed via open networking, thus introducing significant security risks. As wide area network is a very heterogeneous open network, all job and data transmissions have to be protected against, for example, data manipulation or deletion and data theft. Access control of a distributed environment is necessary. It is demonstrated here how to manage a VDE – virtual computing center with user-defined security policies, which are different from the security management rules of underlying Grid infrastructures. The following basic concepts and technologies are applied to the security model of distributed environments.

Public Key Cryptography

Public Key Cryptography is used to encrypt messages that are transferred via the distributed system. The message sender has a key pair, a private key and a public key. The sender encrypts messages with his private key and sends encrypted messages to the receiver. The receiver gets encrypted messages and then decrypts the messages.

Digital Signature

Digital Signature is used to sign the information. It assures the recipient of the information that the information has not been tampered with during the transfer. The sender computes the *hash* of the message with some algorithm, uses private key to encrypt the *hash*, and attaches it to the message. To verify the signed message after it is received, the receiver of the message computes the *hash* of the message with the same hashing algorithm as the sender and decrypts the encrypted *hash* that the sender attaches to the message with the sender's public key. If the newly-computed *hash* and the decrypted *hash* match, it proves that the message signed has not been changed during the transfer.

Certificate

A certificate contains vital information to identify and authenticate a user or a service. A certificate includes four primary pieces of information:

- a subject name, which identifies the user or the object that the certificate represents,
- the public key belonging to the subject,
- the identity of a Certificate Authority (CA) that has signed the certificate, and
- the digital signature of the named Certificate Authority.

Mutual Authentication and OpenSSL

Mutual authentication refers that two parties authenticate each other's identity. The mutual authentication is described as follows:

1. The first party (**A**) sends his certificate to the second party (**B**). This certificate informs **B** who **A** claims to be, what **A**'s public key is, and what Certificate Authority is being used in the certificate.
2. **B** firstly makes sure that the certificate is valid by checking the Certificate Authority's digital signature. After **B** has checked out **A**'s certificate, **B** must make sure that **A** really

is the person declared in the certificate. **B** generates a random message and sends it to **A**, asking **A** to encrypt it.

3. **A** encrypts the message using his private key, and sends it back to **B**.
4. **B** decrypts the message using **A**'s public key. If this maps the original random message, then **B** knows that **A** is who he declares. Now **B** trusts **A**'s identity.
5. The same operation should happen in reverse to reach a successful Mutual Authentication.

The OpenSSL project is a collaborative effort to develop a robust, commercial-grade, full-featured, and open source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.

In cryptography, X.509 [174] is an international standard for Public Key Infrastructure (PKI). X.509 specifies standard formats for public key certificates and a certification path validation algorithm. In the virtual computing center, the security control employs X.509 certificates, which are generated by OpenSSL toolkit.

8.3.2 Public Key Infrastructure of virtual computing center

A UNICORE Public Key Infrastructure (PKI) is set up to organize the security of a virtual computing center. The Public Key Infrastructure consists of the following entities:

- Certificate Authority (CA),
- *gateway* certificates on each UNICORE *gateway*,
- *NJS* certificates for distributing sub-jobs to different *Vsite*s, and
- Client Certificates for all accepted UNICORE users.

To build a customized access control for the virtual computing center, user level security control with *openssl* [98] could be deployed by the administrator of the virtual computing center. The following steps are required:

- Build Certification Authority for the virtual computing center
  A Certificate Authority (CA) contains a CA public key and CA private key. The public key is distributed to CA users. To build a CA for the virtual computing center, for example, use the command such as:
  ```
  openssl req -config openssl.cnf \
  -new -x509 -keyout cakey.pem -out cacert.pem
  ```

- Create certificate for users, UNICORE *gateway* and *NJS*
  Every user of a virtual computing center needs a user public key and a user private key to access UNICORE *gateway* and *NJS*. To create user private key and public key request, a command is used for example:
  ```
  openssl req -config openssl.cnf \
  -new -keyout userkey.pem -out userreq.pem
  ```
  Then the user sends the public certificate request to the Certificate Authority and the public key is then signed by Certificate Authority. The UNICORE *gateway* and the *NJS* process certificates in the same way.

- Deploy access control for the virtual computing center
  The Certificate Authority public key is set as a trusted certificate of the UNICORE *gateway* and the *NJS* to control the resources. The administrator demands the user public key and maps it to a local Unix account in the UUDB, here "*unicore-worker*".

  The users generate a *keystore* with user public key and private key in the UNICORE client side. Thus, the user can access the computing resource inside the virtual computing center via the *keystore*.

8.4 Parallel satellite image processing: a test case

8.4.1 Parallel satellite image processing

This section introduces the test case – parallel satellite image processing. Remotely sensed imagery is becoming a growing source of information in many applications, for example, urban planning, disaster monitoring, and all different kinds of mapping tasks. One of the key issues for all these applications is the ground cover classification, i.e. the raw satellite data have to be analyzed to determine regions with homogenous spectral characteristic and to thematic maps [65]. Ground cover classification aims to exact differentiated classes or themes, land use and land cover categories, from raw remotely sensed digital satellite data.

The Improved Splitting-and-Merging Cluster (ISMC) algorithm [111] is a hybrid classification method for automatic clustering of satellite-observed scenes. It uses a partial clustering algorithm augmented by a hierarchical split-and-merge step at each iteration and dynamically computes the image-specific split-and-merge thresholds. The ISMC algorithm is a computationally intensive application thus demanding extensive CPU cycles.

A distributed ISMC algorithm (D-ISMC in short) [65] has therefore been developed to benefit from computing resources in the distributed environment. The D-ISMC algorithm employs the master-slave paradigm (see also Fig. 8.2). The master process divides the image into multiple sub-images, namely *cluster*s and allocates them to slave processes. The slave processes execute the ISMC algorithm then return the classification results to the master process. Communication is intensive between the master process and the slave processes, for task distribution

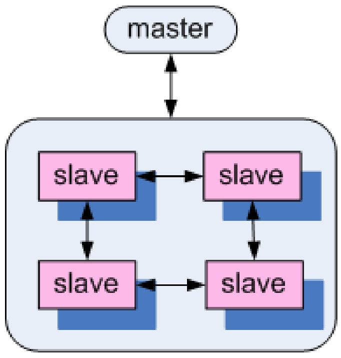


Figure 8.2: Master-slave paradigm of D-ISMC algorithm

and result collection. Communication also exists between the slave processes themselves for exchange of *cluster* boundary information. The communication between processes are implemented via message passing paradigm. The D-ISMC application is thus a tightly coupled distributed application.

The D-ISMC algorithm is shown in Algorithm 1. Firstly the number of involved processes and their ranks are initialized. The rank of master task is set to 0 and the slaves are assigned with positive ranks. Each task gets its own role depending on its rank, then the corresponding function (master task or slave task) is invoked.

Algorithm 1 The D-ISMC algorithm

```
1    BEGIN
2    determine the number of tasks;
3    determine the rank of the task;
4    IF (task id = 0) THEN
5        master task;
6    ELSE
7        slave task;
8    ENDIF
9    END
```

The master task is shown in Algorithm 2. The master task initialize itself with the following steps:

1. read the image size,
2. calculate the size of each task,
3. compute the initial cluster center, and

4. send the task sizes and initial cluster center to corresponding tasks.

The master task then enters the splitting-and-merging part. In this part, the master task exchanges the necessary information with slave tasks and collects the results from the slave tasks to identify the cluster to be split and the clusters to be merged. When the convergence condition is accessed the master sends the finalization message to all the slave tasks.

Algorithm 2 Master task of the D-ISMC algorithm

```
1    BEGIN
2    initialize tasks and assign them to slaves;
3    WHILE (no convergence) DO
4        send cluster information to all slaves;
5        split clusters and remove empty clusters;
6        run k-means algorithm;
7        merge cluster and remove empty clusters;
8        run k-means algorithm;
9       receive results and compute new cluster centers;
10    ENDWHILE
11    send cluster information to all slaves;
12    END
```

The slave task is shown in Algorithm 3. Slave tasks receive the initialization information from the master task. The slave task allocates each pixel in the Image Block (IB) to the nearest cluster center then return the results to the master task.

Algorithm 3 Slave task of the D-ISMC algorithm

```
1    BEGIN
2    get Image Block (IB) information from the master task;
3    receive cluster centers from master task;
4    set all elements to nearest cluster center;
5    compute the new cluster centers;
6    return the newly computed cluster information;
7    END
```

8.4.2 Executing the application in the virtual computing center from UNICORE client

To run the application in the virtual computing center, UNICORE client side should be prepared:

- UNICORE client package should be installed in the user side.
- User side security configuration
  The user generates a user private key and the public key request from the UNICORE client GUI. The user then sends the public key request to the Certificate Authority and gets the signed public key. From the UNICORE client GUI, the user imports the Certificate Authority public key and the *keystore* generated from the user public key and private key pair. The user also requires the UNICORE *gateway* contact information, *gateways.xml*. The user thus can access the virtual computing center managed by UNICORE servers.

The application requires a MPI environment in the UNICORE server. The administrator of the virtual computing center could handle this by running MPI installation scripts inside UNICORE *Vsite* virtual machines or just copying a pre-compiled MPI package to the UNICORE *Vsite* virtual machine.

8.4.3 Test results

Table 8.1 shows the test results. The test is executed in a virtual computing center, which contains 1 master node and 8 slave nodes. The the master algorithm is executed on the master node and the slave algorithm is executed on slave nodes.

Table 8.1: Test results of distributed satellite image processing

Image ID	**size** (**X** × **Y** pixel)	**Tr**(S_b)	**Tr**(S_w)	**T** (s) (8 VMs)	**T** (s) (1 VM)	**T** (s) (1 real machine)
A1	517×472	7539984	16677393	5	28	26
A2	934×500	11827331	23247625	8	53	51
A3	512×512	10473713	19062779	6	36	34
A4	704×503	11526723	18108646	7	46	43
A5	271×329	10865262	17881856	3	7	6
A6	429×436	45339384	12930916	5	26	21
A7	801×867	18371372	23529951	11	76	70

Three qualitative measurements are used here to determine the effectiveness of the clustering procedure:

- $Tr(S_b)$ – trace of the between cluster scatter matrix S_b ,
- $Tr(S_w)$ – trace of the within cluster scatter matrix S_w, and
- T – the execution time of the D-ISMC algorithm.

Figure 8.3 gives an example of 6-band LANDSAT satellite image. Figure 8.4 shows the classification result of Figure 8.3 with the D-ISMC algorithm. There are 4 main classes, shown as red, yellow, blue and green, standing for water area, soil area, forest area and mountain area respectively.

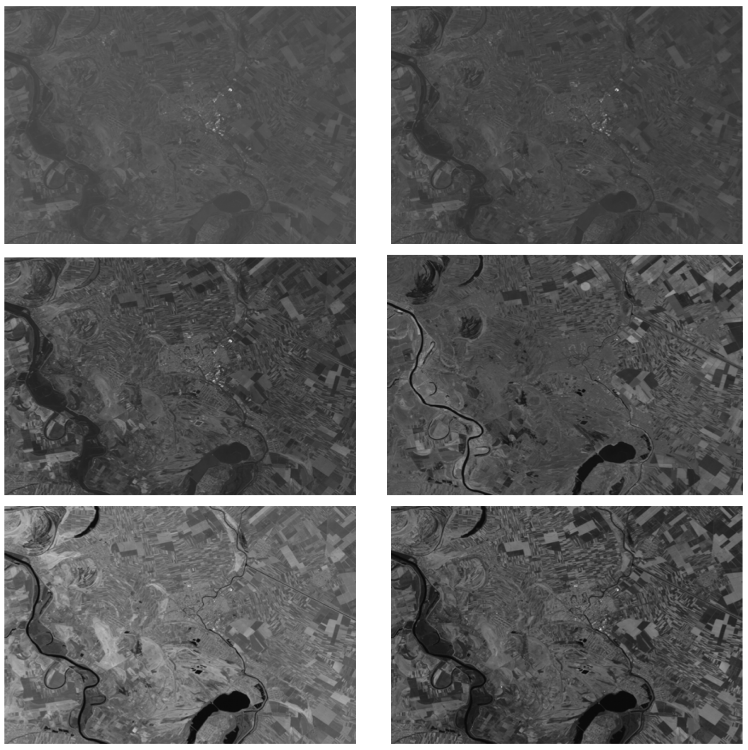


Figure 8.3: 6-band LANDSAT image

8.4.4 Discussion

The satellite image processing is suitable to be executed the virtual computing center:

- The satellite image processing application is a tightly coupled application. Therefore it is better to execute the application on resources, such as a compute center, where high performance communication networking could be found.

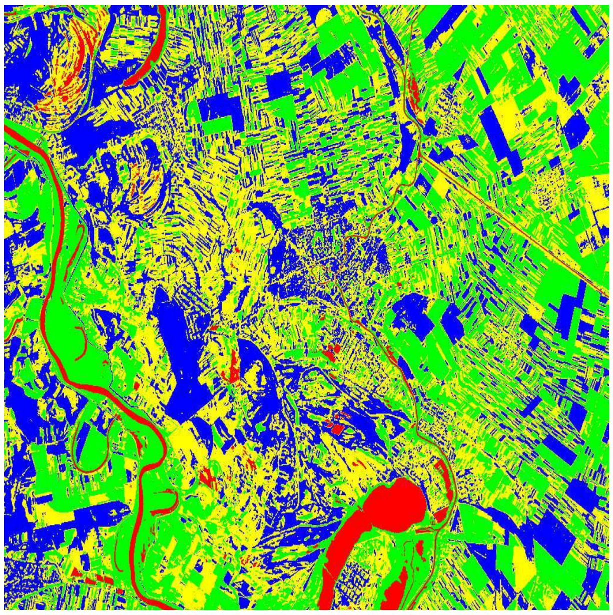


Figure 8.4: Processing result of the 6-band LANDSAT image

- The application domain of satellite image processing requires some special workbenches, such as Linux distributions and image processing libraries. The software environment could be easily implemented by letting application users themselves construct the computing environment, the virtual computing center.
- The satellite image processing application is a data driven application. The compute codes are lightweight while satellite images are only available on some data centers. In the traditional compute model, users always move data to compute resources. This paradigm is not desirable when processing large data set. The virtual computing center paradigm brings a efficient way – to transfer compute codes to data sources.

8.5 Summary

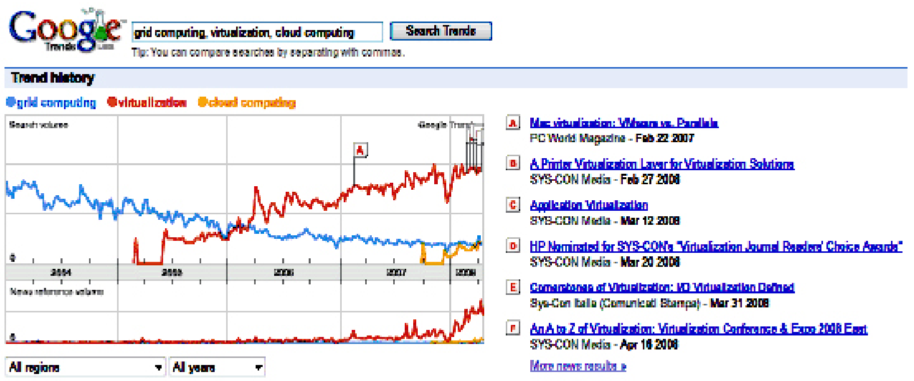
This chapter discusses an interesting VDE use scenario: computing center provides a number of virtual processors and users build a virtual computing center on them. A virtual computing center is constructed with a number of virtual machine disk images, UNICORE as a management middleware, the access control policies and various user level software packages. The builder of a virtual computing center, also acting as the administrator of the virtual computing center, thus provides computing environments to users. This scenarios demonstrate how a VDE is customized with use-defined access control, which is different from that of underlying Grid security mechanisms. An engineering application, parallel satellite image processing, is tested in the virtual computing center to justify the methodology and the implementation.

Chapter 9

Scientific Cloud Computing: an Innovative Computing Paradigm

9.1 Introduction

The Cloud computing, which was coined in late of 2007, currently emerges as a hot topic due to its abilities to offer flexible dynamic IT infrastructures, QoS guaranteed computing environments and configurable software services. As reported in the Google trends shown in Figure 9.1, the Cloud computing (the blue line), which is enabled by virtualization technology (the yellow line), has already outpaced the Grid computing [40] (the red line).



A Mac virtualization: VMware vs. Parallels
PC World Magazine - Feb 22 2007

B A Printer Virtualization Layer for Virtualization Solutions
SYS-CON Media - Feb 27 2008

C Application Virtualization
SYS-CON Media - Mar 12 2008

D HP Nominated for SYS-CON's "Virtualization Journal Readers' Choice Awards"
SYS-CON Media - Mar 20 2008

E Cornerstones of Virtualization: I/O Virtualization Defined
Sys-Con Italia (Comunicati Stampa) - Mar 31 2008

F An A to Z of Virtualization: Virtualization Conference & Expo 2008 East
SYS-CON Media - Apr 16 2008

More news results »

Figure 9.1: Cloud computing in Google trends

A computing Cloud could contain a number of virtual machines and provide users with integrated computing platforms. Therefore a computing Cloud, in nature, could be deemed as an extension of a Virtual Distributed Environment (VDE), which is discussed in the foregoing chapters. This chapter evolves the VDE concept to the Cloud computing paradigm.

There are still no widely accepted definitions for the Cloud computing albeit the Cloud computing practice has attracted much attention. Several reasons lead to this situation:

- Cloud computing involves researchers and engineers from various backgrounds, e.g., Grid computing, software engineering and database. They work on the Cloud computing from different viewpoints.
- Technologies which enable the Cloud computing are still evolving and progressing, for example, Web 2.0 and Service Oriented Computing.
- Existing computing Clouds still lack large scale deployments and killer applications, which would finally justify the concept of Cloud computing.

This chapter presents the Cumulus project, which is intended to build a scientific Cloud for data centers by merging existing Grid infrastructures with new Cloud technologies. Based on the experience of our implementation, this chapter attempts to introduce the concept of Cloud computing: definition, functionalities, enabling technologies and typical applications. This chapter also gives a comparison of Grid computing and Cloud computing.

9.2 Anatomy of Cloud Computing

9.2.1 Definition of Cloud computing

Cloud computing is becoming one of the next IT industry buzz words: users move out their data and applications to the remote "Cloud" and then access them in a simple and pervasive way. This is again a central processing use case. Similar scenario occurred around 50 years ago: a time-sharing computing server served multiple users. Until 20 years ago when personal computers came to us, data and programs were mostly located in local resources. Certainly currently the Cloud computing paradigm is not a recurrence of the history. 50 years ago we had to adopt the time-sharing servers due to limited computing resources. Nowadays the Cloud computing comes into fashion due to the need to build complex IT infrastructures. Users have to manage various software installations, configuration and updates. Computing resources and other hardware are prone to be outdated very soon. Therefore outsourcing computing platforms is a smart solution for users to handle complex IT infrastructures.

At the current stage, the Cloud computing is still evolving and there exists no widely accepted definition. Based on our experience, we propose an early definition of Cloud computing as follows:

A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing platforms on demand, which could be accessed in a simple and pervasive way.

9.2.2 Functional aspects of Cloud computing

Conceptually, users acquire computing platforms or IT infrastructures from computing Clouds and then run their applications inside. Therefore, computing Clouds render users with services to access hardware, software and data resources, thereafter an integrated computing platform as a service, in a transparent way:

- HaaS: Hardware as a Service
  Hardware as a Service was coined possibly in 2006. As the result of rapid advances in hardware virtualization, IT automation and usage metering & pricing, users could buy IT hardware, or even an entire data center, as a pay-as-you-go subscription service. The HaaS is flexible, scalable and manageable to meet your needs [16]. Examples could be found at Amazon EC2 [126], IBM's Blue Cloud project [92], Nimbus [147], Eucalyptus [133] and Enomalism [132].

- SaaS: Software as a Service
  Software or an application is hosted as a service and provided to customers across the Internet. This mode eliminates the need to install and run the application on the customer's local computers. SaaS therefore alleviates the customer's burden of software maintenance, and reduces the expense of software purchases by on-demand pricing.

  An early example of the SaaS is the Application Service Provider (ASP) [128]. The ASP approach provides subscriptions to software that is hosted or delivered over the Internet. Microsoft's "Software + Service" [165] shows another example: a combination of local software and Internet services interacting with one another. Google's Chrome browser [138] gives an interesting SaaS scenario: a new desktop could be offered, through which applications can be delivered (either locally or remotely) in addition to the traditional Web browsing experience.

- DaaS: Data as a Service
  Data in various formats and from multiple sources could be accessed via services by users on the network. Users could, for example, manipulate the remote data just like operate on a local disk or access the data in a semantic way in the Internet.

  Amazon Simple Storage Service (S3) [127] provides a simple Web services interface that can be used to store and retrieve, declared by Amazon, any amount of data, at any time, from anywhere on the Web. The DaaS could also be found at some popular IT services, e.g., Google Docs [139] and Adobe Buzzword [125]. ElasticDrive [131] is a distributed remote storage application which allows users to mount a remote storage resource such as Amazon S3 as a local storage device.

Based on the support of the HaaS, SaaS and DaaS, the Cloud computing in addition can deliver the Infrastructure as a Service (IaaS) for users. Users thus can on-demand subscribe to their favorite computing platforms with requirements of hardware configuration, software installation and data access demands. Figure 9.2 shows the relationship between the services. The Google App Engine [137] is an interesting example of the IaaS. The Google App Engine enables users to build Web applications with Google's APIs and SDKs across the same scalable systems, which power the Google applications.

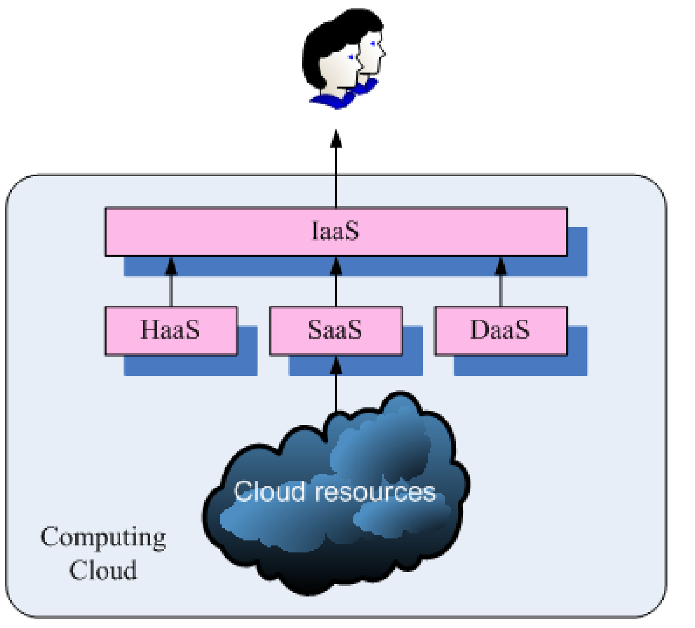


Figure 9.2: Cloud functionalities

9.2.3 Why is Cloud computing distinct?

The Cloud computing distinguishes itself from other computing paradigms, like Grid computing [40], Global computing [33], Internet Computing [71] in the following aspects:

- User-centric interfaces
  Cloud services should be accessed with simple and pervasive methods. In fact, the Cloud computing adopts the concept of Utility computing. In other words, users obtain and employ computing platforms in computing Clouds as easily as they access a traditional public utility (such as electricity, water, natural gas, or telephone network). In detail, the Cloud services enjoy the following features:

  - The Cloud interfaces do not force users to change their working habits and environments, e.g., programming language, compiler and operating system. This feature differs Cloud computing from Grid computing as Grid users have to learn new Grid commands & APIs to access Grid resources & services.

  - The Cloud client software which is required to be installed locally is lightweight. For example, the Nimbus Cloudkit client size is around 15MB.
  - Cloud interfaces are location independent and can be accessed by some well established interfaces like Web services framework and Internet browser.

- On-demand service provisioning
  Computing Clouds provide resources and services for users on demand. Users can customize and personalize their computing environments later on, for example, software installation, network configuration, as users usually own administrative privileges.

- QoS guaranteed offer
  The computing environments provided by computing Clouds can guarantee QoS for users, e.g., hardware performance like CPU speed, I/O bandwidth and memory size.

  The computing Cloud renders QoS in general by processing Service Level Agrement (SLA) with users – a negotiation on the levels of availability, serviceability, performance, operation, or other attributes of the service like billing and even penalties in the case of violation of the SLA.

- Autonomous System
  The computing Cloud is an autonomous system and it is managed transparently to users. Hardware, software and data inside clouds can be automatically reconfigured, orchestrated and consolidated to present a single platform image, finally rendered to users.

- Scalability and flexibility
  The scalability and flexibility are the most important features that drive the emergence of the Cloud computing. Cloud services and computing platforms offered by computing Clouds could be scaled across various concerns, such as geographical locations, hardware performance, software configurations. The computing platforms should be flexible to adapt to various requirements of a potentially large number of users.

9.2.4 Enabling technologies behind Cloud computing

A number of enabling technologies contribute to Cloud computing, several state-of-the-art techniques are identified here:

- Virtualization technology
  Virtualization technologies partition hardware and thus provide flexible and scalable computing platforms. Virtual machine techniques, such as VMware [175] and Xen [8], offer virtualized IT-infrastructures on demand. Virtual network advances, such as VPN [30], support users with a customized network environment to access Cloud resources. Virtualization techniques are the bases of the Cloud computing since they render flexible and scalable hardware services.

- Orchestration of service flow and workflow
  Computing Clouds offer a complete set of service templates on demand, which could be composed by services inside the computing Cloud. Computing Clouds therefore should be able to automatically orchestrate services from different sources and of different types to form a service flow or a workflow transparently and dynamically for users.

- Web service and SOA
  Computing Cloud services are normally exposed as Web services, which follow the industry standards such as WSDL [171], SOAP [163] and UDDI [148]. The services organization and orchestration inside Clouds could be managed in a Service Oriented Architecture (SOA). A set of Cloud services furthermore could be used in a SOA application environment, thus making them available on various distributed platforms and could be further accessed across the Internet.

- Web 2.0
  Web 2.0 is an emerging technology describing the innovative trends of using World Wide Web technology and Web design that aims to enhance creativity, information sharing, collaboration and functionality of the Web [24]. Web 2.0 applications typically include some of the following features/techniques:

  - CSS to sperate of presentation and content,
  - Folksonomies (collaborative tagging, social classification, indexing & social tagging),
  - Semantic Web technologies,
  - REST, XML and JSON-based APIs,
  - Innovative Web development techniques such as Ajax,
  - XHTML and HTML markup,
  - Syndication, aggregation and notification of Web data with RSS or Atom feeds,
  - Mashups, merging content from different sources, client- and server-side,
  - Weblog publishing tools,
  - Wiki to support user-generated content, and
  - Tools to manage users' privacy on the Internet.

  The essential idea behind Web 2.0 is to improve the interconnectivity and interactivity of Web applications. The new paradigm to develop and access Web applications enables users access the Web more easily and efficiently. Cloud computing services in nature are Web applications which render desirable computing services on demand. It is thus a natural technical evolution that the Cloud computing adopts the Web 2.0 technique.

- World-wide distributed storage system
  A Cloud storage model should foresee:

- A network storage system, which is backed by distributed storage providers (e.g., data centers), offers storage capacity for users to lease. The data storage could be migrated, merged, and managed transparently to end users for whatever data formats. Examples are Google File System [45] and Amazon S3 [127]. A Mashup [103] is a Web application that combines data from more than one source into a single integrated storage tool. The SmugMug [164] is an example of Mashup, which is a digital photo sharing Web site, allowing the upload of an unlimited number of photos for all account types, providing a published API which allows programmers to create new functionality, and supporting XML-based RSS and Atom feeds.
- A distributed data system which provides data sources accessed in a semantic way. Users could locate data sources in a large distributed environment by the logical name instead of physical locations. Virtual Data System (VDS) [169] is good reference.

• Programming model
Users drive into the computing Cloud with data and applications. Some Cloud programming models should be proposed for users to adapt to the Cloud infrastructure. For the simplicity and easy access of Cloud services, the Cloud programming model, however, should not be too complex or too innovative for end users.

The MapReduce [20, 21] is a programming model and an associated implementation for processing and generating large data sets across the Google worldwide infrastructures. The MapReduce model firstly involves applying a "map" operation to some data records – a set of key/value pairs, and then processes a "reduce" operation to all the values that shared the same key. The Map-Reduce-Merge [182] method evolves the MapReduce paradigm by adding a "merge" operation. Hadoop [140] is a framework for running applications on large clusters built of commodity hardware. It implements the MapReduce paradigm and provides a distributed file system – the Hadoop Distributed File System. The MapReduce and the Hadoop are adopted by recently created international Cloud computing project of Yahoo!, Intel and HP [17, 134].

9.3 Cloud computing: State of the Art

Numerous projects within industry and academia have already started, for example the RESERVOIR project [162] – an IBM and European Union joint research initiative for Cloud computing, Amazon Elastic Compute Cloud [126], IBM's Blue Cloud [92], scientific Cloud projects such as Nimbus [147], Stratus [166] and OpenNEbula [155]. HP, Intel Corporation and Yahoo! Inc. recently announced the creation of a global, multi-data center, open source Cloud computing test bed for industry, research and education [17]. This section reviews related Cloud computing projects.

9.3.1 OpenNEbula

The OpenNEbula (former GridHypervisor) [155] is a virtual infrastructure engine that enables the dynamic deployment and re-allocation of virtual machines in a pool of physical resources. The OpenNEbula system extends the benefits of virtualization platforms from a single physical resource to a pool of resources, decoupling the server, not only from the physical infrastructure but also from the physical location.

The OpenNEbula contains one frontend and multiple backends. The frontend provides users with access interfaces and management functions. The backends are installed on Xen servers, where Xen hypervisors are started and virtual machines could be backed. Communications between frontend and backends employ SSH. The OpenNEbula gives users a single access point to deploy virtual machines on a locally distributed infrastructure.

9.3.2 Amazon Elastic Compute Cloud

The Amazon Elastic Compute Cloud (EC2) [126] provides Web service interfaces which deliver resizable compute capacities to users in the computing Cloud. Users could rent and configure virtual machines and data storage easily and use the computing capacities remotely.

In more detail with Amazon EC2 users can:

- create an Amazon Machine Image (AMI) which contains applications, libraries, data and associated configuration settings, or use Amazon pre-configured, templated images to start up and run immediately,
- upload the AMI into Amazon Simple Storage Service (S3). The Amazon S3 provides a safe, reliable and fast repository to store users' virtual machine images and data objects,
- employ Amazon EC2 Web service to configure security and network access,
- choose the type of instance that users want to execute, and
- start, shutdown, and monitor the instances of user's AMI using Web service APIs.

9.3.3 Nimbus

Based on the Globus Virtual Workspace Service, a Cloudkit named Nimbus [147] is developed to build scientific Clouds. Nimbus provides a free, open source infrastructure for remote deployment and management of virtual machines, allowing users to:

- Create computing Clouds,
- Deploy "one-click" auto-configuring virtual clusters,

- Serve clients that are compatible with the Amazon EC2 service,
- Integrate virtual machines on a set of resources already configured to manage jobs (i.e., already using a batch scheduler like PBS), and
- Interface to Amazon EC2 resources.

9.3.4 Eucalyptus

The Eucalyptus (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems) project [133] from UCSD is an open-source software infrastructure for implementing Elastic/Utility/Cloud computing using computing clusters and/or workstation farms. The Eucalyptus enjoys several interesting features, such as compatible interfaces with Amazon's EC2, simple installation, deployment and management, support virtual private networks for users.

The Eucalyptus provides Amazon EC2 compatible interfaces. The Eucalyptus implements mediator services to translate users' requirements to the Cloud backends: the client-side API translator and the Cloud controller. Multiple Rock clusters serve as backends of the Eucalyptus Cloud, which demand cluster controllers and node controllers.

9.4 Cloud Computing and Grid computing: a Comparative Study

This section is devoted to compare the Cloud computing and the Grid computing in various aspects, such as definitions, infrastructures, middleware and applications. It is interesting to develop computing Clouds on the existing Grid infrastructures to get advantages of Grid middleware and applications. We make a comparative study in following aspects:

- Definition

  Grid computing, oriented from high performance distributed computing, aims to share distributed computing resource for remote job execution and for large scale problem solving. The Grid computing emphasizes the resource side by making huge efforts to build an independent and complete distributed system. Cloud computing provides user-centric functionalities and services for users to build customized computing environments. Cloud computing, which is oriented towards the industry, follows an application-driven model.

- Infrastructure

  Grid infrastructure enjoys the following features:

  - Grid infrastructure in nature is a decentralized system, which spans across geographically distributed sites and lack central control;
  - Grid infrastructure normally contains heterogeneous resources, such as hardware/software configurations, access interfaces and management policies;

  On the contrary, from the viewpoint of users, computing Clouds operate like a central compute server with single access point. Cloud infrastructures could span several computing centers, like Google and Amazon, in general contain homogeneous resources, operated under central control.

- Middleware

  Grid computing has brought up full-fledged middleware, for example, Unicore [101], Globus Toolkit [135] and gLite [46]. The Grid middleware could offer basic functionalities like resource management, security control and monitoring & discovery. The Grid community has established well-defined industry standards for Grid middleware, e.g., WSRF [172].

  The middleware for Cloud computing, or the Cloud operating system, is still underdeveloped and lacks of standards. A number of research issues remain unsolved, for example, distributed virtual machine management, Cloud service orchestration, and distributed storage management.

- Accessibility and application

  Grid computing has the ambitious objective to offer dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. However inexperienced users still find difficulties to adapt their applications to Grid computing. Furthermore, it is not easy to get a performance guarantee from computational Grids. Cloud computing, on the contrary, could offer customized, scalable and QoS guaranteed computing environments for users with an easy and pervasive access.

  Grid computing has gained numerous successful stories in many application fields. A recent famous example is LCG project [95] to process huge data generated from the Large Hadron Collider (LHC) at CERN. Although Amazon EC2 has obtained a beautiful success, more killer applications are still needed to justify the Cloud computing paradigm.

Based on the forementioned discussion, it could be concluded that Grid computing has established well organized infrastructures, middleware and application experience. Cloud computing recently outpaces Grid computing because it could offer computing environments with desirable features, such as QoS guarantee and computing environment customization. It would be a good solution to build Cloud computing services on Grid infrastructure and middleware, thus providing users with satisfying services. Next section discusses a scientific Cloud computing prototype based on the existing Grid infrastructure.

9.5 Cumulus: a Scientific Computing Cloud

9.5.1 Overview

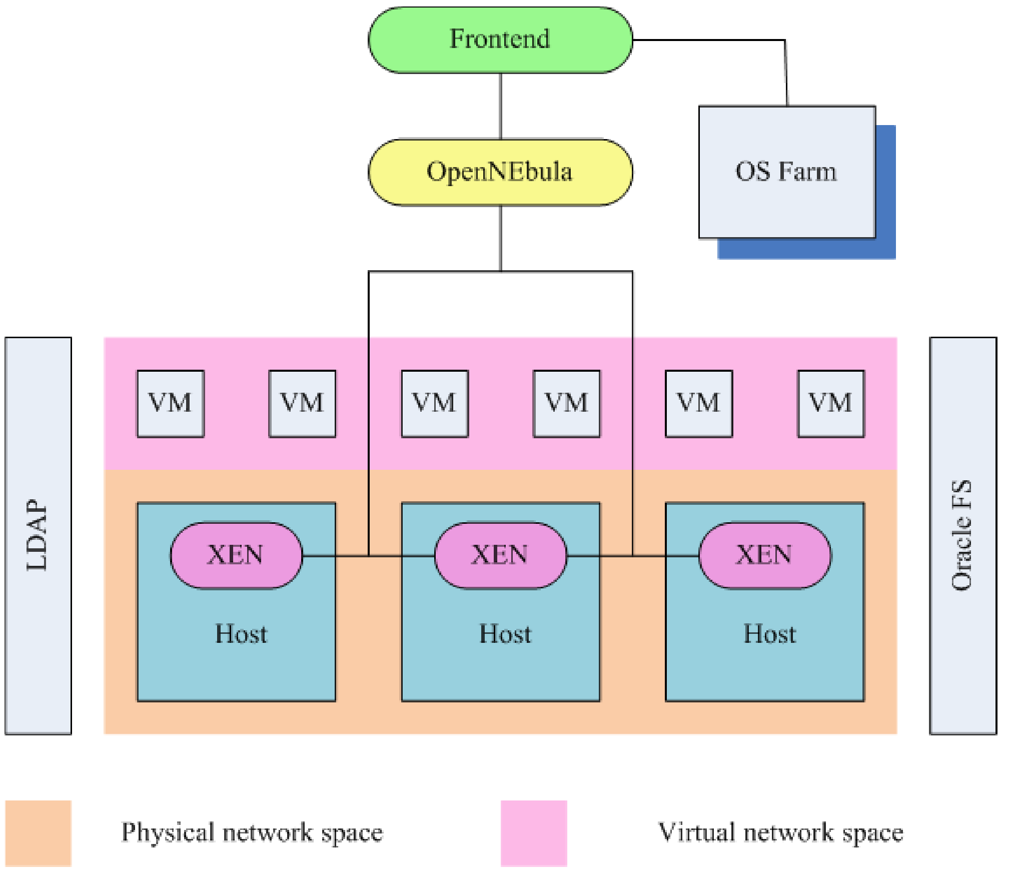


Figure 9.3: Cumulus architecture

The Cumulus project is an on-going Cloud computing project in Steinbuch Computing Center (SCC), which intends to provide virtual computing platforms for scientific and engineering applications. The Cumulus project currently is running on distributed IBM blade servers with Oracle Linux and the Xen hypervisors. We design the Cumulus system in a layered architecture (see also Figure 9.3):

- The Cumulus frontend service resides on the access point of a computing center and accepts users' requirements of virtual machine operations.
- The OpenNEbula works as Local Virtualization Management System (LVMS) on the Cumulus backends. The OpenNEbula communicates with the Cumulus frontend via SSH or XML-RPC.

- The OpenNEbula contains one frontend and multiple backends. The OpenNEbula frontend communicates to its backends and the Xen hypervisors on the backends via SSH for virtual machine manipulation.
- OS Farm
  The OS Farm [99] is a Web server for generating and storing Xen virtual machine images and Virtual Appliances. The OS Farm is used in the Cumulus system as a tool for virtual machine template management.

This design pattern could bring the scalability and keep the autonomy of compute centers:

- The Cumulus frontend does not depend on any specific LVMS, we use a third party software – the OpenNebula as an example here.
- Compute centers inside computing Clouds could define their own resource management policies such as network solution or virtual machine allocation.
- The Cumulus frontend can also delegate users' requirements to other computing Clouds.

9.5.2 Cumulus frontend: re-engineering of the Globus Virtual Workspace Service

The Globus Virtual Workspace Service (GVWS) [56] contains two pieces of software: the GVWS frontend and the GVWS control agents.

The GVWS frontend receives the virtual machine requirements and distributes them to various backend servers via SSH. The GVWS control agent is installed on every backend server and communicates with the the Xen server in order to deploy a virtual workspace.

In principal the GVWS frontend could be employed as a Cloud frontend service. However some limitations of the GVWS are identified :

- Computer centers in general run their own specific LVMS, like OpenNEbula or VMware Virtual Infrastructure, to manage their local infrastructures. However, the GVWS demands to install the workspace control agents directly on the backend servers. This use scenario provided by the GVWS lacks of the generality.
- The GVWS provides three network settings: the *AcceptAndConfigure* mode, the *AllocateAndConfigure* mode and the *Advisory* mode. However users generally do not care about network configurations for virtual machines. Users only demand the virtual machine network address or host name so that they can finally access the virtual machines across the network. We suggest that the network configurations and solutions are transparent to Cloud users and they only controlled by the Cloud backends. This paradigm would be more reasonable considering that the Cloud infrastructure might span across multiple network domains.

The GVWS is re-engineered to adapt it as the Cumulus frontend service. This comprises the following steps:

- Extend the GVWS frontend and remove the control agent, thus allowing the GVWS to work with various virtual machine hypervisors and LVMS, such as OpenNEbula, VMware server and VMware Virtual Infrastructure.
- Support a new networking solution – the *forward* mode. In the *forward* mode, users input no network configuration information; the backend servers allocate the IP address for the virtual machine and return it to users.

9.5.3 OpenNEbula as the Cumulus backend

The OpenNEbula is used to manage the local distributed servers, which provides virtual machine deployment. Currently the OpenNEbula employs the NIS (Network Information System) to manage a common user system and the NFS (Network File System) for virtual machine image management. However it has been widely recognized that the NIS has a major security flaw: it leaves the users' password file accessible by anyone on the entire network. The NFS has some minor performance issues and it does not support concurrency. To employ the OpenNEbula in a more professional way, we merged the OpenNEbula with some modern secure infrastructure solutions like the Lightweight Directory Access Protocol (LDAP) [58] and the Oracle Cluster File System (OCFS) [157] or the Andrew File System (AFS) [152].

The LDAP is an application protocol for querying and modifying directory services over TCP/IP. A directory is a set of objects with similar attributes organized in a logical and hierarchical manner. In the OpenNEbula scenario, the OpenLDAP software is used to implement the user's directory. A common user, *oneadmin*, is authorized on the OpenNEbula frontend and all the OpenNEbula backends.

The OpenNEbula frontend and the backends share directories using the OCFS. The shared space is used to store virtual machine images and to allow the OpenNEbula system to behave as single virtual machine hypervisor.

9.5.4 Communication between the Cumulus frontend and the OpenNEbula

The communication between the Cumulus frontend and the OpenNEbula service could be either SSH (Secure Shell) or XML-RPC: the Cumulus frontend requires virtual machine operations for example by remotely invoking the OpenNEbula commands via SSH or by sending XML-RPC messages via the OpenNEbula APIs. For other LVMS, like VMware Infrastructure, communication methods should be developed and plugged into the Cumulus frontend.

- Communication with SSH

  Originally, the GVWS frontend issues its commands to its backends via SSH. Therefore the first attempt to communicate the Cumulus frontend with the OpenNEbula via SSH. However the SSH communication is not supported by current OpenNEbula implementation. The OpenNebula Command Line interface (CLI) is the interface used when issuing SSH commands. It identifies the virtual machines running on the OpenNEbula with an identification number when a virtual machine is started in the context of the OpenNEbula. As a consequence this identification number should be returned to the Cumulus frontend for later use, for example, to pause, resume and reboot the just created virtual machine. The OpenNEbula uses the terminal as the standard output to print the identification number. The SSH program, following Unix conventions, can only get the return code after it is executed. It cannot get the command line output, which is needed to be parsed to obtain virtual machine's identification number. Therefore, the SSH communication is not supported in the current implementation context.

- Communication with XML-RPC

  The OpenNEbula exposes the XML-RPC interfaces for users to access its services. The OpenNebula's features could therefore be accessed from the client programs. This method in some aspects is superior to the previous CLI implementation since easy recovery of the virtual machine's identification number is possible. Upon creation of virtual machines, the XML-RPC call returns a parameter with the identification number of the created virtual machine. Also all other functions return the status of the executed commands and therefore a better control and interaction with the OpenNebula server are possible.

9.5.5 OS Farm for virtual machine image management

The Cumulus can generate virtual machine images for users by using the OS Farm. The OS Farm [99] is an application written in Python to generate virtual machine images and virtual appliances used with the Xen VMM. Two methods are provided for users to connect to the OS Farm server:

- Web interface

  The Web interface provides various methods to enable users to request virtual machine images. The easiest method is the simple request which allows users to select image class and architecture. The advanced request gives the possibility to add *yum* packages to the virtual machine image. A drop down menu allows the user to choose packages from a list of predefined *yum* repositories. Using asynchronous Java Script and XML, a further list is returned which allows users to select the corresponding *yum* packages. Support for request via XML descriptions is also available. An image description can be uploaded through the Web interface of the OS Farm server as an XML file. A sample XML description is shown below:

```
<image>
    <name>myvm</name>
    <class>slc4_old</class>
    <architecture>i386</architecture>
    <package>emacs</package>
    <package>unzip</package>
    <group>Base</group>
    <group>Core</group>
</image>
```

- HTTP interface

  The OS Farm also provides the HTTP interface. For example, users can use the *wget* to access the HTTP interface of the OS Farm server. Below is a sample usage of the HTTP interface with the *wget* command:

  ```
  wget http://www.fzk.de/osfarm/create?name=&transfer=http&
  class=slc_old&arch=i386&filetype=.tar&group=core&group=base
  ```

The images in the OS Farm are generated in layers which can be cached. The generation of a virtual machine image is divided into three layers or stages. Figure 9.4 shows the layers of virtual machine image generation:

- The *core* layer is a small functional image with a minimal set of software which is either required to run the image on a VMM or required to satisfy higher level software dependencies.
- The *base* layer provides some additional software needed in order to satisfy requirements for *virtual appliances*.
- the *image* layer is on top of *core* layer and provides user-defined software in addition to the core software.
- A subclass of image is *virtual appliance*, which is an image with an extra set of rules aimed to allow the image to satisfy requirements of the deployment scenario.

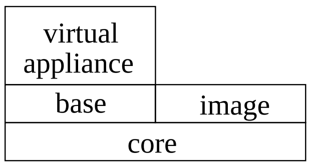


Figure 9.4: Layers of the OS Farm image generation

An OS Farm client is embedded in the Cumulus frontend to generate virtual machine images. When Cumulus users send virtual machine requirements to the Cumulus frontend, the frontend invoke the OS Farm client, which thereafter sends HTTP requests to the OS Farm server. The virtual machine images generated by the OS Farm are transferred to the Cumulus virtual machine repository, which is implemented via the OCFS. Then users could manipulate virtual machines after they are stared by the Cumulus backends (see also Figure 9.5).

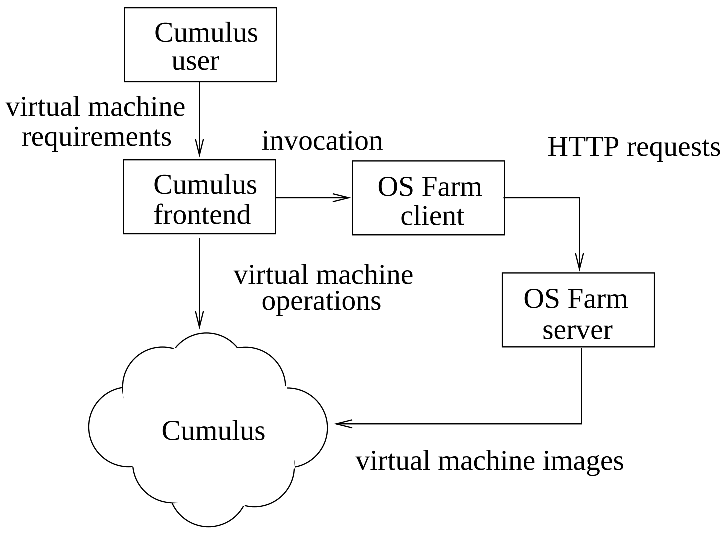


Figure 9.5: Using OS Farm in the Cumulus

The OS Farm client is implemented as follows:

1. The client gets virtual machine parameters from the Cumulus frontend, such as virtual machine name, transfer protocol, virtual machine class name, virtual machine architecture, virtual machine image file format, and image generation layer. An example of these parameters are “myvm, http, slc4_old, i386, .tar, core”

2. The client then constructs a HTTP request and send it to the OS Farm server. For example, a *wget* URL could be built like this:

```
wgeturl = osfarmserver +
"/create?name=" + name
+ "& transfer =" + transferprotocol
+ "& class =" + class
+ "& arch =" + arch
+ "& filetype =" + filetype
```

3. Finally the client obtains the virtual machine image from the OS Farm server and stores it in the Cumulus virtual machine repository. A connection is formed between the OS

Farm server and the client with the help of *java.net.URL* class. It is demanded to read and write the coming data from the OS Farm server simultaneously due to the large size of the virtual machine image files (200 MB – 2GB). It is not possible to store such large image files in the default heap space of the Java virtual machine, which is normally around 128 MB.

9.5.6 Networking solution

We provide a new networking solution – the *forward* mode. Users do not have to specify the network configuration. The backend servers could implement the network solution for the incoming virtual machine requests. The *forward* mode could play an important role when the Cloud infrastructure span across multiple network domains.

When virtual machine requirements arrive at the Cumulus frontend, the frontend just forwards the requirements to the Cumulus backend. The backend sets the virtual machine network configuration and returns the network configurations to the frontend. The frontend thereafter again forwards the configurations to users. Finally users could access the virtual machine via the network configurations, normally the IP address or the host name of the virtual machine. Specifically, three methods to allocate virtual machine networking are implemented in the Cumulus backend:

- Lease IP addresses from an IP address pool
  In this solution, a file named "ipPool" stores, as deducted from the name, a pool of IP addresses. In this context, when the frontend submits the virtual machine to the backend, the first available IP address in the file will be taken and be parsed as a Xen parameter. Since the OpenNebula (via Xen) already supports setting a virtual machine IP address, it is available that on virtual machine's creation the virtual machine is configured with the provided IP address. Other networking parameters, such as gateway, subnet mask, DNS and broadcast, can be specified in OpenNebula's configuration files. This solution already provides functions to avoid reusing an already consumed IP address and assures that a new IP address is used for every new virtual machine. The administrator of the backend is only required to declare in the file the available IPs.

- Identify a virtual machine with the host name
  In this implementation, the host name parameter uniquely identifies a virtual machine. When the frontend sends a virtual machine to the backend, the backend set the host name for the virtual machine, then returns it to the frontend for later reference. However, the OpenNebula is not yet capable of asking the Xen server to set a host name. It is demanded to ask the OpenNebula to directly submit this parameter to the Xen hypervisor by setting the OpenNebula's "RAW" variable. This variable is used to directly pass to the Xen server, on virtual machine's creation, any additional Xen parameter users would like to add. Moreover, this solution permits no further configuration of any additional files, since

the implementation already generates a unique host name on virtual machine's creation time. The backend can return the host name to users for later reference.

- Get IPs from a central DHCP server
  In this implementation, when the frontend sends virtual machines to the backend, the backend sets the networking item in the configuration file with "DHCP". Then the virtual machine is started, listens to the central DHCP server and gets the network configurations. However, the problem here is how to get the IP address of the started virtual machine and returns it to the frontend. Since every virtual machine has its own unique MAC address, one solution is to query the DHCP server to obtain the virtual machine's IP address by providing the MAC address. On successful reply, the backend would forward the IP address to the frontend, so that user could access the virtual machine. A possible drawback related to this solution is its dependence on certain specific networking configuration. Depending on administrator's policies of the backend, some queries to the DHCP server might be banned.

9.6 Summary

This chapter reviews recent advances of Cloud computing and presents our views on the Cloud computing: definition, key features and enabling technologies. This chapter also presents a comparison of the Grid computing and the Cloud computing. A prototype of scientific Cloud – the Cumulus project is also discussed. The Cumulus project aims to build a scalable scientific Cloud: it could adopt to various virtual machine hypervisors as backend, furthermore delegates users' requirements to other Clouds.

Chapter 10

Conclusion and Future Work

10.1 Conclusion

Grid computing now emerges as a practical and efficient computing paradigm. It supports large scale resource sharing, work collaboration and provides nontrivial Qualities of Service for innovative applications. With the aid of Grid middleware, standard Grid services and protocols, users could execute their work on Grid resources. Though it brings huge benefits of Grid computing, this use methodology however delivers numerous challenges to the Grid community:

- Grid users may demand a personalized computing environment provided by remote Grid resources, such as hardware/software configuration and user privileges. Considering that a large number of users on computational Grids, various resource requirements and security control methods, it is difficult for current Grid system to offer a customized computing environment to each Grid user.

- Computational Grids aim to render unpeered Qualities of Service (QoS) to Grid users. Current Grid system employs traditional multi-programmed model, where multiple Grid users and local users compete for Grid resources simultaneously. This model is inefficient for QoS support.

- Grid concept is overloaded by endowing Grid middleware with too many functionalities, resulting that the deployment of Grid systems and the execution of Grid applications become complex tasks. This scenario limits the further progress of Grid computing.

Virtual machine and virtual environment recently gain research interests as they can realize various advantages, e.g., performance guarantee, on-demand creation and customization of computing environments. The Doctoral thesis therefore provides a new philosophy of Grid usage. Instead of enabling Grid applications by executing them directly on Grid resources, in the new methodology customized virtual machines and virtual environments are provided to

Grid users. With the virtual environments provided, Grid users could find desired computing environments, for example, requisite hardware/software, user privileges and QoS. Resource providers are also released from tedious, onerous and also inevitable tasks of resource management.

This thesis gets the following important scientific achievements:

1. Virtual environment for Grid computing – an innovative philosophy for Grid computing
   The thesis identifies current issues and challenges facing Grid computing and then proposes a philosophy of virtual environment for Grid computing. In contrast to traditional methodology that maps Grid jobs directly to Grid resources, the Doctoral work proposes a new methodology: to provide Virtual Machines (VM) and Virtual Distributed Environments (VDE) for Grid jobs.

2. Grid Virtualization Engine
   The Grid Virtualization Engine (GVE) is an innovative lightweight Grid middleware to assist construction of VMs & VDEs on Grid infrastructures. The Doctoral thesis presents the design and the implementation of Grid Virtualization Engine in detail. Current implementation of Grid Virtualization Engine consists of two services: Information Service and Virtualization Service.

3. Information Service for monitoring distributed virtual machines
   The new Information Service provides interfaces to access resource information of distributed VMs and VDEs. It is implemented with lightweight Java agents and integrated into existing Grid middleware, such as Globus Toolkit.

4. Virtualization Service for operating distributed virtual machines
   The new Virtualization Service supports functions of VM and VDE provision and management. It is designed and implemented in a hierarchical and distributed flavor, and it can work with various virtual machine hypervisors thus offering a uniform access layer for VMs and VDEs.

5. Grid Workflow in a Virtual Box
   The virtual environment detaches the user computing environment from Grid resources. Users therefore can customize their computing environments, operating systems, software packages and application programs then put them together into a Virtual Box. In our example, SUSE operating system, Globus Toolkit, Condor, VDS, and the application are packaged into a Grid Workflow Virtual Box and then is submitted to the virtual resources at the remote computing center. This novel usage demonstrates easy control on a VDE: resources, middleware, data and applications are packaged together into Virtual Box and this could be submitted, executed and migrated easily. A test case of CMS workflow is implemented in the use methodology of Grid Workflow in a Virtual Box.

6. Dynamically building a virtual e-Science infrastructure
   In the traditional Grid model, users must acquire Grid resources before job submission.

We organize an e-Science infrastructure that dynamically demands virtual machines as computing resources via the Grid Virtualization Engine, installs necessary software libraries and packages, e.g., ActiveBEPL and GridSAM, and executes e-Science applications on the virtual machines. This novel use case justifies that a VDE could be dynamically built, deployed and executed on-demand. A bio-sequence alignment application is implemented in the virtual e-Science infrastructure.

7. Virtual computing center
   Users could build a virtual computing center, deploy the virtual computing center on distributed virtual machines via the Grid Virtualization Engine, then steer it with common Grid middleware, e.g. UNICORE. This novel use scenario instantiates that users can define their own management policies inside the VDE, e.g., X.509 certificate offered by *openssl*, which is independent to the underlying Grid infrastructure. The virtual computing center paradigm employs a satellite image processing application as an example.

8. Cloud computing: definition and experience
   Benefitting from the virtualization technologies in distributed computing, Cloud computing emerges as an innovative computing paradigm to enable building flexible IT infrastructures on demand. This Doctoral thesis brings the pioneering work on Cloud computing: definition of Cloud computing and building a scientific computing Cloud – the Cumulus project.

The thesis could therefore conclude:

1. Methodology of providing VMs and VDEs for users could solve current research problems and limitations of Grid computing;

2. Our implementation of a lightweight Grid middleware, Grid Virtualization Engine, could help build virtual environments for Grid computing;

3. Exciting solutions for VDEs are expected in various application communities and use scenarios.

10.2 Future work

The philosophy of virtual environment for Grid computing could be refined and improved in various aspects:

- Virtualization on data and network resource
  In the thesis, only compute resource is virtualized, in term of virtual machine, and provided to Grid infrastructures. Virtual network and virtual data should also be considered to create a fully virtual environment. It is not only desirable, but also necessary since data

and networks as well as compute resources are indispensable for distributed computing environments.

- Virtual application
  As presented in Figure 1.1 and Section 2.1.1, a layered Grid system model contains, top to bottom, in brief, application layer, middleware layer and resource layer. Philosophically, this Doctoral work carries out virtualization from bottom to top: virtualize resources for applications. The Grid Virtualization Engine is built in the middleware level. It is therefore interesting to make virtualization from top to bottom: virtualize applications for resources. A virtual application could be created by wrapping normal applications with standard interfaces: e.g., software requirement, resource resource requirement, invoking interface and control parameters. With these standard interfaces, virtual applications could be very easy to compile, deploy and execute on various types of Grid resources.

To build a productional Grid Virtual Engine, the following aspects should be investigated further:

- Security issues when managing distributed virtual machines and VDEs
  In the methodology of virtual environment for Grid computing, a Virtual Organization (VO) is mapped to a Virtual Distributed Environment (VDE). Therefore user authentication and authorization inside the VO is left to the user level. Security control occurs when the administrator of a VO requires virtual machines from distributed computing centers by communicating with the Grid Virtualization Engine. In this thesis, security issues are simply handled by adopting user-password authentication. Possible solution may exist via re-use of current Gird security infrastructure, which is installed in computing centers.

- Virtual network
  Virtual networks connect virtual machines and multiplex physical networks. Virtual network should present an overlay network topology over TCP/IP level. Customization of virtual networks are required for VDEs, such as IP address, network topology. Therefore, user level virtual network should be supported in the Grid Virtualization Engine, for example Peer to Peer (P2P) network and Virtual Private Network.

- Information model
  Current management of virtual machines employs extended GLUE schema. To build a productional information service for distributed virtual machines, industry standard for information retrieval should be followed. It is a proper solution to suggest OGF for extension of the GLUE schema to adopt the virtual machine concept.

- Manage distributed virtual machine images
  To construct a virtual machine or a VDE, virtual machine disk images and software packages are required. There should be a data management system to index, transfer and store the distributed virtual machine images and various software packages. Virtual

Data System might be a good candidate as it can manage data files transfer and map the abstract virtual machine images to some specific physical locations.

- Uniform interfaces of Virtual Machine Monitors
  One of current encumbrances to employ virtual machine for Grid computing is that different VMMs provide various user interfaces and programming interfaces. For example, VMware ESX server offers Java API for its Web service interface, VMware Server support C and Perl API with TCP/IP communication and Xen API is currently under development. Therefore, an industry standard should be defined for virtual machine interfaces.

- Re-use of existing Grid middleware
  It is a wise choice to re-use current Grid middleware to construct a productional Grid Virtualization Engine. Key implementation of information service, resource management and security control from current Grid middleware could be adopted in the Grid Virtualization Engine. The Grid Virtualization Engine requires not only full productional functions but also lightweight implementation, scalability and portability. UNICORE thus is a good candidate in that:

  - The UNICORE is implemented in modularity structure using pure Java language, which guarantees the scalability and portability.
  - The core components of UNICORE are lightweight implemented with simple interfaces for access and management.
  - The UNICORE complies with the industry standards, e.g., WSRF from OGF.

Index

Bibliography

[1] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, Mi. Zhao, L. Zhu, and X. Zhu. From virtualized resources to virtual computing Grids: the In-VIGO system. *Future Generation Comp. Syst.*, 21(6):896–909, 2005.

[2] GT 4.0 Information Services: aggregator Framework [URL]. http://www.globus.org/toolkit/docs/4.0/info/aggregator/, access on Nov. 2007.

[3] K. Al-Saqabi, S. A. Sarwar, and K. Saleh. Distributed gang scheduling in networks of heterogenous workstations. *Computer Communications*, 20(5):338–348, 1997.

[4] G. Allen, T. Goodale, M. Russell, E. Seidel, and J. Shalf. *Classifying and Enabling Grid applications*, chapter in Grid computing: making the global infrastructure a reality, pages 601–614. John & Wiley, F. Berman and G. Fox and T. Hey edition, 2003.

[5] W. Andreoni and A. Curioni. New advances in chemistry and materials science with CPMD and parallel computing. *Parallel Computing*, 26(7-8):819–842, 2000.

[6] A.Szalay, P. Kunszt, A. Thakar, J. Gray, and D. Slutz. Designing and mining multiterabyte astronomy archives: the sloan digital sky survey. In *Proceedings of the ACM International Conference on Management of Data*, pages 451–462, 2000.

[7] R. Balzer. Process virtual machine. In *Proceedings of the 7th International Workshop on Software Process*, pages 37–40, 1991.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 164–177, New York, U. S. A., Oct. 2003.

[9] R. Buyya, D. Abramson, and J. Giddy. Research from the trenches: Nimrod-G resource broker for service-oriented Grid computing. *IEEE Distributed Systems Online*, 2(7), 2001.

[10] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. GridFlow: workflow management for Grid computing. In *Proceedings of the 3rd IEEE International Symposium on Cluster Computing and the Grid*, pages 198–205, 2003.

[11] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. Sprenkle. Dynamic virtual clusters in a Grid site manager. In *Proceedings of the 12th International Symposium on High Performance Distributed Computing*, pages 90–103, 2003.

[12] J. Chen and Y. Yang. Adaptive selection of necessary and sufficient checkpoints for dynamic verification of temporal constraints in Grid workflow systems. *ACM Transactions on Autonomous and Adaptive Systems*, 2(2), 2007.

[13] J. Chen and Y. Yang. Multiple states based temporal consistency for dynamic verification of fixed-time constraints in Grid workflow systems. *Concurrency and Computation: Practice and Experience*, 19(7):965–982, 2007.

[14] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and scalability of a replica location service. In *Proceedings of the 13th International Symposium on High-Performance Distributed Computing*, pages 182–191, 2004.

[15] K. Chu, L. Di, and P. Thornton. Introduction of Grid computing application projects at the NASA earth science technology office. In *Proceedings of the First International Conference on Advances in Grid and Pervasive Computing*, pages 289–298, 2006.

[16] Here comes HaaS [URL].
http://www.roughtype.com/archives/2006/03/here_comes_haas.php/,
access on June 2008.

[17] Global Cloud computing test bed [URL].
http://www.hp.com/hpinfo/newsroom/press/2008/080729xa.html/, access on July 2008.

[18] D. Culler, J. P. Singh, and A. Gupta. *Parallel computer architecture: a hardware/software approach*. Morgan Kaufmann, August 1998.

[19] K. Czajkowski, I. Foster, Nicholas T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Proceedings of International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[20] J. Dean. Mapreduce and other building blocks for large-scale distributed systems at google. In *Proceedings of the USENIX Annual Technical Conference*, 2007.

[21] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[22] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto Grid environments. *J. Grid Comput.*, 1(1):25–39, 2003.

[23] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[24] Web 2.0 definition [URL].
http://en.wikipedia.org/wiki/web_2/, access on June 2008.

[25] D. Diachin, L. Freitag, D. Heath, J. Herzog, W. Michels, and P. Plassmann. Remote engineering tools for the design of pollution control systems for commercial boders. *International Journal Supercomputer Applications*, 10(2):208–218, 1996.

[26] J. B. Drake and I. T. Foster. Introduction to the special issue on parallel computing in climate and weather modeling. *Parallel Computing*, 21(10):1539–1544, 1995.

[27] e-Science definition [URL].
http://en.wikipedia.org/wiki/e-science, access on Nov. 2007.

[28] I. Foster (ed), J. Frey (ed), S. Graham (ed), S. Tuecke (ed), K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling stateful resources with Web services v. 1.1. Technical report, Global Grid Forum, Mar. 2004.

[29] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price. Grid service orchestration using the Business Process Execution Language (BPEL). *J. Grid Comput.*, 3(3-4):283–304, 2005.

[30] B. Gleeson etc. A framework for IP based virtual private networks. RFC2764, The Internet Engineering Task Force, Feb. 2000.

[31] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr., and H. Truong. ASKALON: a tool set for cluster and Grid computing. *Concurrency – Practice and Experience*, 17(2-4):143–169, 2005.

[32] T. Fahringer, S. Pllana, and A. Villazón. A-GWL: Abstract Grid Workflow Language. In *Proceedings of International Conference on Computational Science*, pages 42–49, 2004.

[33] G. Fedak, C. Germain, V. Néri, and F. Cappello. XtremWeb: a generic global computing system. In *Proceedings of the 1st IEEE International Symposium on Cluster Computing and the Grid*, pages 582–587, 2001.

[34] R. J. O. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A case for Grid computing on virtual machine. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 550–559, 2003.

[35] Enable Grid for e-Science (EGEE) project [URL].
http://www.eu-egee.org/, access on Nov. 2007.

[36] OASIS: Organization for the Advancement of Structured Information Standards [URL].
http://www.oasis-open.org/, access on Nov. 2007.

[37] Java Architecture for XML Binding (JAXB) [URL].
http://java.sun.com/developer/technicalarticles/webservices/jaxb/,
access on Jan. 2007.

[38] I. Foster. What is the Grid? a three point checklist. *GRIDToday*, June 2002.

[39] I. Foster, J. Geisler, B. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-WAY metacomputing experiment. *Concurrency – Practice and Experience*, 10(7):567–581, 1998.

[40] I. Foster and C. Kesselman. *The Grid: blueprint for a new computing infrastructure*. Morgan Kaufmann, 1998.

[41] I. Foster and C. Kesselman. The Globus project: a status report. *Future Generation Comp. Syst.*, 15(5-6):607–621, 1999.

[42] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid – enabling scalable virtual organizations. *Int'l J. of Supercomp. App.*, 15(3):200–222, August 2001.

[43] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The Open Grid Services Architecture (OGSA), version 1.0. Technical report, Global Grid Forum (GGF), Jan. 2005.

[44] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: a computation management agent for multi-institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.

[45] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 29–43, 2003.

[46] gLite [URL]. http://glite.web.cern.ch/glite/, access on Sep. 2008.

[47] A. S. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Wide-area computing: resource sharing on a large scale. *IEEE Computer*, 32(5):29–37, 1999.

[48] Global Grid Forum GIS Working Group. Grid information services for resource sharing. Technical report, Global Grid Forum, July 2001.

[49] Grid Interoperation Now Community Group. Experiences from interoperation scenarios in production Grids. Technical report, Open Grid Forum, Feb. 2007.

[50] S. Hariri and A. Varma. High-performance distributed computing: promises and challenges. *Concurrency – Practice and Experience*, 5(4):233–238, 1993.

[51] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006.

[52] D. E. Irwin, J. S. Chase, L. E. Grit, A. R. Yumerefendi, D. Becker, and K. Yocum. Sharing networked resources with brokered leases. In *Proceedings of the 2006 USENIX Annual Technical Conference*, pages 199–212, 2006.

[53] X. Jiang and D. Xu. SODA: a service-on-demand architecture for application service hosting utility platforms. In *Proceedings of the 12th International Symposium on High-Performance Distributed Computing*, pages 174–183, 2003.

[54] X. Jiang and D. Xu. VIOLIN: virtual internetworking on overlay infrastructure. In *Proceedings of the 2nd International Symposium of Parallel and Distributed Processing and Applications*, pages 937–946, 2004.

[55] K. Keahey, K. Doering, and I. Foster. From sandbox to playground: dynamic virtual environments in the Grid. In *Proceedings of the 5th International Workshop on Grid Computing*, pages 34–42, 2004.

[56] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: achieving quality of service and quality of life in the Grid. *Scientific Programming*, 13(4):265–275, 2005.

[57] L. Kleinrock. UCLA to be 1st station in nationwide computer network. Press release, University of California, Los Angeles, July 1969.

[58] V. A. Koutsonikola and A. Vakali. LDAP: framework, practices, and trends. *IEEE Internet Computing*, 8(5):66–72, 2004.

[59] G. V. Laszewski and M. Hategan. Workflow concepts of the Java CoG Kit. *J. Grid Comput.*, 3(3-4):239–258, 2005.

[60] C. Lee, C. Kesselman, and S. Schwab. Near real-time satellite image processing: metacomputing in CC++. *IEEE Computer Graphics and Applications*, 16(4):79–84, 1996.

[61] C. Lee and D. Talia. *Grid programming models: current tools, issues and directions*, chapter in Grid computing: making the global infrastructure a reality, pages 555–578. John Wiley & Sons Inc., F. Berman and G. Fox and T. Hey edition, December 2002.

[62] A. Lenstra. Factoring integers using the Web and the number field sieve. In *Proceedings of International Forum on Multimedia and Information Security*, pages 93–113, 1995.

[63] T. Lindholm and F. Yellin. *Java virtual machine specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[64] M. Litzkow, M. Livny, and M. W. Mutka. Condor – a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, 1988.

[65] B. Liu. Ground cover classification of satellite imagery on parallel architectures. Master's thesis, School of Computing Engineering, Nanyang Technological University, Singapore, 2004.

[66] K. Marzullo, M. Ogg, A. Ricciardi, A. Amoroso, F.A. Calkins, and E. Rothfus. NILE: widearea computing for high energy physics. In *Proceedings of 7th ACM SIGOPS European Workshop*, pages 49–54, 1996.

[67] K. M. McCann, M. Yarrow, A. D. Vivo, and P. Mehrotra. ScyFlow: an environment for the visual specification and execution of scientific workflows. *Concurrency and Computation: Practice and Experience*, 18(10):1155–1167, 2006.

[68] D. A. Menascé and E. Casalicchio. QoS in Grid computing. *IEEE Internet Computing*, 8(4):85–87, 2004.

[69] P. Messin, S. Brunett, D. Davis, T. Gottschalk, D. Curkendall, L. Ekroot, and H. Siegel. Distributed interactive simulation for synthetic forces. In *Proceedings of the 11th International Parallel Processing Symposium*, 1997.

[70] C. Metz. The latest in virtual private networks: Part I. *IEEE Internet Computing*, 7(1):87–91, 2003.

[71] M. Milenkovic, S. H. Robinson, R. C. Knauerhase, D. Barkai, S. Garg, V. Tewari, T. A. Anderson, and M. Bowman. Toward Internet distributed computing. *IEEE Computer*, 36(5):38–46, 2003.

[72] M. Mowafi, N. Jiang, T. Caudell, W. Shu, and M. Wu. Real-time transmission of stereo images over the access Grid. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 633–636, 2002.

[73] Germany national Grid initiative [URL]. http://www.dgrid.de/, access on Nov. 2007.

[74] E. Newcomer and G. Lomow. *Understanding SOA with Web services*. Addison-Wesley, 2004.

[75] T. Nguyen and C. Plumejeaud. An integration platform for metacomputing applications. In *Proceedings of the International Conference on Computational Science*, pages 474–483, 2002.

[76] J. Nieplocha and R. Harrison. Shared memory NUMA programming on the I-WAY. In *Proceedings of 5th IEEE Symposium on High Performance Distributed Computing*, pages 340–349, 1996.

[77] K. Nozaki, T. Akiyama, S. Shimojo, S. Maeda, and H. Tamagawa. Integration of computational fluid dynamics and computational aero acoustics on Grid for dental applications. In *Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems*, pages 517–522, 2005.

[78] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for Redundant Arrays of Inexpensive Disks (RAID) . In *Proceedings of the ACM International Conference on Management of Data*, pages 109–116, 1988.

[79] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für Instrumentelle Mathematik, Bonn Universität, 1962.

[80] C. Potter, R. Brady, P. Moran, C. Gregory, B. Carragher, N. Kisseberth, J. Lyding, and J. Lindquist. EVAC: a virtual environment for control of remote imaging. *IEEE Computer Graphics and Applications*, 16(4):62–66, 1996.

[81] C. Potter, Z. P. Liang, C. Gregory, H. Morris, and P. Lauterbur. Toward a neuroscope: a real-time system for the evaluation of brain function. In *Proceedings of 1st IEEE International Conference on Image Processing*, pages 25–29, 1994.

[82] L. Pouchard, A. Woolf, and D. Bernholdt. Data Grid discovery and semantic Web technologies for the earth sciences. *Int. J. on Digital Libraries*, 5(2):72–83, 2005.

[83] APAC Grid project [URL]. http://grid.apac.edu.au/, access on Nov. 2007.

[84] Cactus project [URL]. http://www.cactuscode.org/, access on Nov. 2007.

[85] China CROWN project [URL]. http://www.crown.org.cn/en/, access on Nov. 2007.

[86] Earth System Grid project [URL].
https://www.earthsystemgrid.org/, access on Nov. 2007.

[87] France Grid5000 project [URL]. http://www.grid5000.fr/, access on Nov. 2007.

[88] Fraunhofer Resource Grid project [URL].
http://www.fhrg.fhg.de/index_en.html/, access on Nov. 2007.

[89] GridAnt project [URL].
http://www.globus.org/cog/projects/gridant/, access on Nov. 2007.

[90] GriPhyN project [URL]. http://www.gridphyn.org/, access on Nov. 2007.

[91] GSFL project [URL].
http://www-unix.globus.org/cog/projects/workflow/, access on Nov. 2007.

[92] IBM Blue Cloud project [URL].
http://www-03.ibm.com/press/us/en/pressrelease/22613.wss/,
access on June 2008.

[93] IPG project [URL]. http://www.ipg.nasa.gov/, access on Nov. 2007.

[94] Japan NAREGI Grid project [URL]. http://www.naregi.org/index_e.html, access on Nov. 2007.

[95] LHC Computing Grid project [URL]. http://cern.ch/lhcgrid/, access on Nov. 2007.

[96] MPICH project [URL].
http://www.mcs.anl.gov/research/projects/mpich2/, access on Jan. 2007.

[97] NorduGrid project [URL]. http://www.nordugrid.org/, access on Nov. 2007.

[98] OpenSSL project [URL]. http://www.openssl.org/, access on Nov. 2007.

[99] OS Farm project [URL]. http://cern.ch/osfarm /, access on July 2008.

[100] OSCAR project [URL]. http://cmsdoc.cern.ch/oscar/, access on Nov. 2007.

[101] Unicore project [URL]. http://www.unicore.org/, access on Nov. 2007.

[102] Xen project [URL]. http://www.xen.org/, access on June 2008.

[103] Mashup project[URL].
http://en.wikipedia.org/wiki/mashup_web_application_hybrid/, access on June 2008.

[104] J. Robin and C. Irvine. Analysis of the Intel pentium's ability to support a secure virtual machine monitor. In *Proceedings of the 9th USENIX Security Symposium*, pages 129–144, Denver, Colorado, USA., Aug. 2000.

[105] M. Russell, G. Allen, G. Daues, I. Foster, E.Seidel, J. Novotny, J. Shalf, and G. v. Laszewski. The astrophysics simulation collaboratory: a science portal enabling community software development. *Cluster Computing*, 5(3):297–304, 2002.

[106] P. Ruth, X. Jiang, D. Xu, and S. Goasguen. Virtual distributed environments in a shared infrastructure. *IEEE Computer*, 38(5):63–69, 2005.

[107] S. V. Scott and W. Venters. The practice of e-science and e-social science. In *Virtuality and Virtualization*, pages 267–279, 2007.

[108] L. Seawright and R. MacKinnon. VM/370 – a study of multiplicity and usefulness. *IBM Systems Journal*, 18(1):4–17, 1979.

[109] B. A. Shirazi, K. M. Kavi, and A. R. Hurson, editors. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[110] A. Shoykhet, J. Lange, and P. Dinda. Virtuoso: a system for virtual machine marketplaces. Technical Report NWU-CS-04-39, Northwest University, July 2004.

[111] J. J. Simpson, T. J. McIntire, and M. Sienko. An improved hybrid clustering algorithm for natural scenes. *IEEE Transactions on Geocsience and Remote Sensing*, 38(2):1016–1032, Mar. 2000.

[112] L. Smarr and C. E. Catlett. Metacomputing. *Commun. ACM*, 35(6):44–52, 1992.

[113] J. Smith and R. Nair. *Virtual machines: versatile platforms for systems and processes*. The Morgan Kaufmann, 2003.

[114] T. Smith. and T. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[115] CERN CMS Software and Computing Group. Object oriented reconstruction for CMS analysis. Technical report, CMS Internal Note, CMS-IN 1999/001, 1999.

[116] CERN CMS Software and Computing Group. CMKIN user guide. Technical report, CMS Internal Note, CMS-IN 2004/052Z, 2004.

[117] CMS software [URL]. http://cms.cern.ch/software/cms/, access on Nov. 2007.

[118] GridSAM: Grid Job Submission and Monitoring Web Service [URL]. http://gridsam.sourceforge.net/, access on Nov. 2007.

[119] W. Sudholt, K. Baldridge, D. Abramson, C. Enticott, S. Garic, C. Kondric, and D. Nguyen. Application of grid computing to parameter sweeps and optimizations in molecular modeling. *Future Generation Comp. Syst.*, 21(1):27–35, 2005.

[120] A. I. Sundararaj and P. A. Dinda. Towards virtual networks for virtual machine Grid computing. In *Proceedings of the 3rd Virtual Machine Research and Technology Symposium*, pages 177–190, 2004.

[121] E. G. Talbi and A. Y. Zomaya. Grids in bioinformatics and computational biology. *J. Parallel Distrib. Comput.*, 66(12):1481, 2006.

[122] I. Taylor, M. Shields, I. Wang, and A. Harrison. Visual Grid workflow in Triana. *J. Grid Comput.*, 3(3-4):153–169, 2005.

[123] I. J. Taylor, M. S. Shields, I. Wang, and R. Philp. Distributed P2P computing within Triana: a galaxy visualization test case. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium*, page 16, 2003.

[124] A. Tirado-Ramos, K. Zajac, Z. Zhao, P. M. A. Sloot, G. D. v. Albada, and M. Bubak. Experimental Grid access for dynamic discovery and data transfer in distributed interactive simulation systems. In *Proceeding of the International Conference on Computational Science*, pages 284–294, 2003.

[125] Adobe Buzzword [URL].
http://www.adobe.com/acom/buzzword/, access on Sep. 2008.

[126] Amazon Elastic Compute Cloud [URL].
http://aws.amazon.com/ec2/, access on Nov. 2007.

[127] Amazon Simple Storage Service [URL].
http://aws.amazon.com/s3/, access on Sep. 2008.

[128] Application Service Provider [URL].
http://msdn.microsoft.com/en-us/architecture/aa699384.aspx/,
access on Sep. 2008.

[129] Common Information Model (CIM) [URL].
http://www.dmtf.org/standards/cim/, access on Nov. 2007.

[130] Dsitributed Management Task Force [URL].
http://www.dmtf.org/, access on Nov. 2007.

[131] Elasticdrive Project [URL]. http://www.elasticdrive.com/, access on Sep. 2008.

[132] Enomalism Project [URL]. http://www.enomalism.com/, access on Sep. 2008.

[133] Eucalyptus Project [URL]. http://eucalyptus.cs.ucsb.edu/, access on Sep. 2008.

[134] Global Cloud Computing Research Test Bed Wiki [URL]. http://cloudtestbed.org/, access on Oct. 2008.

[135] Globus Toolkit [URL]. http://www.globus.org/toolkit/, access on Nov. 2007.

[136] Glue Schema V1.3 [URL].
http://glueschema.forge.cnaf.infn.it/spec/v13/, access on Jan. 2007.

[137] Google App Engine [URL].
http://code.google.com/appengine/, access on Sep. 2008.

[138] Google Chrome [URL]. http://www.google.com/chrome/, access on Sep. 2008.

[139] Google Docs [URL]. http://docs.google.com/, access on Sep. 2008.

[140] Hadoop [URL]. http://hadoop.apache.org/, access on Sep. 2008.

[141] Hibernate Mapping Framework [URL].
http://www.hibernate.org/, access on Nov. 2007.

[142] Load Sharing Facility [URL].
http://www.platform.com/products/platform-lsf-family/, access on Nov. 2007.

[143] Microsoft .NET Framework [URL].
http://msdn2.microsoft.com/en-us/netframework/default.aspx/,
access on Nov. 2007.

[144] MPI: Message Passing Interface [URL].
http://www.mpi-forum.org/, access on Nov. 2007.

[145] National Cosmology Grid Porject [URL].
http://www.cosmogrid.ie/, access on Oct. 2008.

[146] Native Virtualization [URL]. http://www.virtualiron.com/, access on Nov. 2007.

[147] Nimbus Project [URL].
http://workspace.globus.org/clouds/nimbus.html/, access on June 2008.

[148] OASIS UDDI Specification [URL].
http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm,
access on June 2008.

[149] OASIS Web Services Business Process Execution Language [URL].
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel/,
access on Nov. 2007.

[150] OMG CORBA [URL]. http://www.corba.org/, access on Nov. 2007.

[151] OMII-BPEL [URL]. http://sse.cs.ucl.ac.uk/omii-bpel/, access on Nov. 2007.

[152] Open AFS [URL]. http://www.openafs.org/, access on Sep. 2008.

[153] Open Grid Forum (OGF) [URL]. http://www.ogf.org/, access on Nov. 2007.

[154] Open Science Grid (OSG) [URL].
http://www.opensciencegrid.org/, access on Nov. 2007.

[155] OpenNEbula Project [URL]. http://www.opennebula.org/, access on Apr. 2008.

[156] OpenPBS [URL]. http://www.pbsgridworks.com/, access on Nov. 2007.

[157] Oracle Cluster File System [URL].
http://oss.oracle.com/projects/ocfs/, access on June 2008.

[158] Pegasus CIMOM [URL]. http://www.openpegasus.org/, access on Nov. 2007.

[159] PVM: Parallel Virtual Machine [URL].
http://www.csm.ornl.gov/pvm/, access on Nov. 2007.

[160] Remote Procedure Call (RPC) [URL].
http://www.cs.cf.ac.uk/dave/c/node33.html, access on Nov. 2007.

[161] Replica Location Service (RLS) [URL].
http://www.globus.org/toolkit/docs/4.0/data/rls/, access on Nov. 2007.

[162] Reservoir Project [URL]. http://www-03.ibm.com/press/us/en/pressrelease/23448.wss/, access on June 2008.

[163] Simple Object Access Protocol (SOAP) [URL].
http://www.w3.org/tr/soap/, access on Nov. 2007.

[164] SmugMug [URL]. http://www.smugmug.com/, access on June 2008.

[165] Software + Services [URL].
http://www.microsoft.com/softwareplusservices/, access on June 2008.

[166] Status Project [URL]. http://www.acis.ufl.edu/vws/, access on June 2008.

[167] The Large Hadron Collider Computing Grid [URL].
http://lcg.web.cern.ch/lcg/, access on Nov. 2007.

[168] Unified Modeling Language [URL].
http://www.omg.org/technology/documents/formal/uml.htm,
access on Nov. 2007.

[169] Virtual Data System [URL]. http://vds.uchicago.edu/, access on Nov. 2007.

[170] Web Service [URL]. http://www.w3.org/2002/ws/, access on Nov. 2007.

[171] Web Service Description Language (WSDL) [URL].
http://www.w3.org/tr/wsdl/, access on Nov. 2007.

[172] Web Service Resource Framework [URL].
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf,
access on June 2008.

[173] World Wide Web Consortium (W3C) [URL].
http://www.w3.org/, access on Nov. 2007.

[174] X.509 [URL]. http://www.itu.int/rec/t-rec-x.509/en/, access on Nov. 2007.

[175] VMware virtualization technology [URL].
http://www.vmware.com, access on Nov. 2007.

[176] L. Wang, W. Cai, B. Lee, B. Schmidt, and W. Jie. Biogrid computing – parallel computing for protein alignment analysis. In *Proceedings of 6th International Conference on High Performance Computing in Asia Pacific Region*, 2002.

[177] L. Wang, W. Jie, and H. Zhu. *State of the arts: workflow system for Grid computing*, chapter in Grid technologies: emerging from distributed architectures to virtual organizations. WIT press, M. Bekakos and A. A. Gravvanis and H. R. Arabnia edition, 2006.

[178] L. Wang and M. Kunze. Organization of CMS benchmarks in VDS workflow on virtual machines. In *Proceedings of the 6th International Conference on Grid and Cooperative Computing*, pages 749–756, 2007.

[179] H. Wenzel. Benchmarking AMD Opteron (AMD64) and Interl (EM64T) systems. Technical report, CMS Internal Note, CMS-IN 2005/012, CMS Software and Computing Group, 2005.

[180] H. Wenzel and S. Langley. Benchmarking dual-core AMD Opteron (AMD64) systems. Technical report, CMS Internal Note, CMS-IN 2005/030, CMS Software and Computing Group, 2005.

[181] A. Whitaker, M. Shaw, and S. Gribble. Scale and performance in the denali isolation kernel. In *Proceedings of the 5th Symposium on Operating System Design and Implementation*, 2002.

[182] H. Yang, A. Dasdan, R. Hsiao, and D. S. Parker Jr. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1029–1040, 2007.

Institut für Angewandte Informatik /
Automatisierungstechnik
Universität Karlsruhe (TH)

Grid computing aims to support flexible, secure, coordinated resource sharing among dynamic groups of individuals, institutions and resources. Despite the fact that numerous progress has been made, some challenges still obstruct the advance of Grid computing, for example, provision of desired user computing environments and Quality of Service (QoS).

Virtualization technology recently emerged as a promising solution for building desirable, manageable and flexible information infrastructures. Various advantages of virtualization have been identified, for instance, performance guarantees and computing environment customization.

This book proposes a new philosophy of Grid usage: provide virtual computing environments for Grid users. The new methodology could solve the problems currently encountered by Grid computing. It supports desired computing environments for users, which provide customized hardware/ software configuration, QoS assurance and manageable functionality. This philosophy also liberates resource providers from the various onerous tasks of resource management. In this book, a lightweight middleware – Grid Virtualization Engine, which renders the functionality to enable virtual environments to be built on Grid infrastructures, has been developed and presented. This book also discusses three typical virtual environments to justify the philosophy: Grid workflow in a Virtual Box, virtual e-Science infrastructure on demand and virtual computing center.

This book advances the Grid vision and Grid usage philosophy by rendering a new methodology for Grid usage: provision of and operation on virtual environments on shared Grid infrastructures.

ISSN: 1614-5267
ISBN: 978-3-86644-310-5

www.uvka.de