Agent-based Manager for Grid Cloud System

By

Osama Hamdy Younis

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

KING ABDULAZIZ UNIVERSITY

JEDDAH – SAUDI ARABIA

Rajab, 1435 H – May, 2014 G

بسم الله الرحمن الرحيم

Agent-based Manager for Grid Cloud System

By Osama Hamdy Younis

A thesis submitted in partial fulfillment of the requirements for degree of Master of Science (Computer Science)

Supervised By
Prof. Dr. Fathy A. Eassa

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

KING ABDULAZIZ UNIVERSITY

JEDDAH – SAUDI ARABIA

Rajab 1435H – May 2014 G

Agent-based Manager for Grid Cloud System

By

Osama Hamdy Younis

This thesis has been approved and accepted in partial fulfillment of the requirements for the degree of Master of Science

EXAMINATION COMMITTEE

	Name	Rank	Field	Signature
Internal Examiner				
External Examiner				
Advisor				

KING ABDULAZIZ UNIVERSITY

Rajab 1435H – May 2014 G

Dedicated to

My parents, my brothers, my sisters and my friends.

ACKNOWLEDGEMENT

At the beginning, praise and gratitude be to ALLAH almighty, without His gracious help, it would have been impossible to accomplish this work. Working on this thesis has been a very interesting and valuable experience to me and I have learned a lot. I want to express my thanks to the people who have been very helpful during the time it took me to finish this thesis.

First, I would like to thank my supervisor Professor *Dr. Fathi E. Al-Borai* who helped me with guidance, supervision, and constructive comments until I completed this work. He is not only my supervisor; he is just like my father. I enjoyed working on this thesis under his supervision, I am grateful to him.

My special gratitude goes to my parents whose love and affection is the source of motivation and encouragement for my studies. I would like to thank all my family, all my sisters and all my brothers for unconditional support, and for being there for me at any time. Special and great thanks go to my mother, the one who deserves to be dedicated all my efforts and achievements. Whatever I say about her is not enough to thank her for the unconditional love and support. Also I am deeply thankful to my brothers and sisters for their love and encouragement.

I would like to express my gratitude to all my colleagues and my friends for useful discussions and prayers, in addition to everyone who asked, supported and encouraged me in any way.

Thank you all…

Osama Younis, May, 2014

ABSTRACT

Nowadays, the processing power of a single system is obviously insufficient for many complex scientific problems that need high computing power to process, and we do not always have the option of using supercomputers or HPCs. Now, cloud infrastructures are being involved in many institutions with different services (e.g. computing, storage, software) provided at different levels of availability, performance, cost and reliability.

With the great advance in software and hardware, cloud-computing systems developed to utilize and benefit from these advances efficiently to solve a broad range of intensive-computing problems. Grid computing, which aggregates several machines' resources, can give a similar power of computing as we can get from a supercomputer, yet with the required quality of service (QoS) that meets the client's needs. This can be achieved through our proposed Manager that aggregates cloud resources (i.e. Computing and Storage) to make a grid of clouds that enhance resource utilization. Because of the heterogeneity of cloud resources in this grid, a platform independent middleware is required for resource management.

In this thesis, we present the design and architecture of an Agent-based Manager for Grid Cloud Systems (AMGCS) using software agents to ensure the independency and the scalability when the numbers of resources and jobs increase. The AMGCS handles IaaS resources and schedules compute-intensive jobs for execution over the available resources according to the QoS criteria. It shows a good performance in executing complex tasks submitted from regular machines, with an optimized task execution and high resource utilization, through the power of grid clouds' capabilities.

مدير مُعتمِد على الوكيل البرمجي لحوسبة شبكة من الأنظمة السحابية

أسامه حمدي يونس عبد الحميد

المستخلص

في الوقت الحاضر، أصبحت قوة معالجة نظام واحد غير كافية للعديد من المشكلات العلمية المعقدة في العلوم والهندسة والتي تتطلب قوة حوسبة عالية لمعالجة مثل هذه المهام، كما أن استخدام "الحاسبات العملاقة" او أجهزة "الحوسبة عالية الأداء" قد لا يكون اختيارًا متاحًا في كل الأحوال. وأصبحت البنى التحتية السحابية تُستخدم في العديد من المؤسسات لتقديم خدمات مختلفة (حوسبة، مساحات تخزين، برامج) وتُقدم على مستويات مختلفة من التكلفة، سرعة الاستجابة، الأداء ومدى التوفر والجاهزية.

مع التقدم الكبير في اجهزة ومكونات الحاسبات والبرمجيات، تم تطوير انظمة الحوسبة السحابية للاستفادة من هذه الخواص بطريقة فعالة لحل المشكلات المعقدة التي تحتاج متطلبات معالجة قوية. الحوسبة الشبكية، والتي تربط مجموعة من موارد الحاسبات ببعضها البعض، تستطيع تحقيق نفس قوة الحوسبة التي تحققها اجهزة "الحوسبة عالية الأداء" مع تحقيق متطلبات جودة الخدمة لاحتياجات المستخدم. يمكن تحقيق ذلك من خلال المدير المُقترَح هنا والذي يقوم بتجميع الخدمات السحابية مثل وحدات المعالجة المركزية ووحدات التخزين لعمل شبكة من الأنظمة السحابية لتسهيل وتعزيز الاستفادة من هذه المصادر. هذه الشبكة تضم مختلف الموارد السحابية والتي تكون غير متجانسة، لذلك نحن بحاجة الى وسيط استقلالي لا يعتمد على منصة (معمارية) او نظم محددة لإدارة هذه الموارد.

في هذه الرسالة، نقدم تصميم ومعمارية مدير مُعتمد على الوكيل البرمجي لحوسبة شبكة من الانظمة السحابية باستخدام الوكيل البرمجي لضمان الاستقلالية وإمكانية التوسع عندما تزداد عدد الموارد والمهام. يستفيد هذا المدير من الموارد المتاحة على شكل "خدمات البُنى التحتية" (IaaS) بطريقة فعّالة لتنفيذ المهام التي تحتاج قوة معالجة عالية وذلك وفقًا لمعايير جودة الخدمة المطلوبة. يُظهر هذا المدير أداءً جيدًا في تنفيذ المهام المعقدة المُرسلة من الأجهزة العادية بالطريقة الأمثل وبدرجة عالية من استغلال الموارد، وذلك من خلال قوة الإمكانات المتوفرة على شبكة الأنظمة السحابية.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AMGCS	Agent-based Manager for Grid Cloud System
VM	Virtual Machine
MPI	Message Passing Interface
QoS	Quality of Service
SLA	Service Level Agreement
SMP	Symmetric Multi-Processing
XML	Extensible Markup Language
GCE	Google Compute Engine
GCS	Google Cloud Storage
AC	Azure Compute
JSON	JavaScript Object Notation
IaaS	Infrastructure-as-a-Service
PaaS	Platform-as-a-Service
SaaS	Software-as-a-Service
REST	Representational state transfer
HTTP	Hypertext Transfer Protocol

Chapter 1

Introduction

1.1 Introduction

The growing need for computational resources to solve large-scale problems leads to the cloud computing approach. Making a grid of cloud computing allows to include a variety of resources like clusters, supercomputers, storage systems, computational kernels and Symmetric Multi-Processors (SMPs) including PCs and workstations, etc. These resources are coupled to be available as a single integrated resource.

The Grid cloud infrastructure can benefit many applications, including distributed supercomputing, high-throughput computing and data exploration. One type of grids is the computational grid, to bring supercomputing power to users by using resources in the network. The emerged Grid Cloud computing is a computing paradigm to solve complex applications in science and engineering, as it involves the combined effective utilization of a cloud resources to achieve high performance computing [1]. To utilize the benefits of the grid clouds, we need an efficient grid cloud management system, a complicated system, as it involves the distributed, heterogeneous and dynamically available resources, as well as handling diverse needs for these resources [2].

The scheduler here is an important module of the system as it is responsible for managing and selecting the geographically available resources, and for scheduling jobs efficiently to meet the user or application requirements, in terms of performance and utilizing resources. Therefore, the acceptability and efficiency of a resource management of computational grid clouds depends on its scheduling strategy and how the process of allocating the needed resources to the job requests is done, and also on how the resource manager is able to determine the availability of specific resources, and mapping jobs to these resources [3].

Hence, the manager needs to achieve the following efficiently: dividing jobs into tasks (if possible), scheduling and monitoring jobs, and sending the proper requests to the associated cloud, all via agents that are communicating together.

Clouds can fulfill a huge amount of computations that cannot be done by the best supercomputers. However, Cloud computing performance can be improved by making sure that all the available resources in the grid cloud are utilized by good QoS algorithm to make sure that most resources are involved in the cloud grid computations under the required criteria. Due to the disparate of job arrival and the inequality of computing capabilities, the resources in one grid cloud may be overloaded (or doesn’t meet the QoS criteria specified) while others in different grid cloud may be available. Therefore, dispatching jobs must be to the right resources to reduce the job average response time and achieve a better resource utilization.

The grid cloud manager will adopt the software agent technology to handle the heterogeneity and interoperability, though; it is hard to build high performance and reliable agents applications that meet the grid cloud requirements.

Here, we present an Agent-based Manger for Grid Cloud System, to manage distributed computational resources in grid clouds by scheduling and providing an efficient way of processing high computing requests, based on software agents using Jade, to adapt the requirements of scalability, robustness and interoperability by the grid cloud system. Jade is an open-source middleware for implementations of multi-agents systems in compliance with FIPA specifications (http://www.fipa.org) [4].

1.2 Thesis Motivation

There are increasing number of cloud service providers, varying in the quality of service provided, and complex tasks are getting increase in fields of science and engineering. We want to take advantage of these clouds' services and utilize them in a proper way. Combining services together from multiple cloud providers will open new directions of computational capabilities, so instead of using costly supercomputers and High Performance Computers we can group Compute services together by creating a grid of clouds. Also with storage services, which are available on different levels from many service providers, would be great if there is a manager that selects the most proper resource that meets our needs.

Compute and Storage services are categorized as Infrastructure-as-a-Service on the cloud, so we built a manager that manages these services and select properly from vast cloud resources available on multiple clouds. Selecting the proper resource helps in executing compute-intensive jobs using cloud resources without the need of HPCs or Supercomputers. Hence, a significant reduce in cost and response time, in addition to a high performance and throughput compared to a single cloud system. This manager achieves high utilization for cloud resources that are aggregated in a grid cloud system.

1.3 Thesis Objectives and Methodology

Many factors affect resource management for grid cloud systems. The main factors are the heterogeneity of resources, scalability degree of the manager and the performance of each cloud in the grid. Heterogeneity and scalability can be solved by building the system to be interoperable and platform-independent using an agent-based architecture to be scalable to large number of resources without affecting the performance. The performance of each cloud can be maintained by using a scheduler agent that distributes load over the available resources, and the system performance can be evaluated depending on resource utilization, waiting and execution time.

In this research, we introduce an Agent-based Manger for Grid Cloud System to manage distributed IaaS resources in the grid clouds based on software Agents, using Jade [4], to adapt scalability, robustness and interoperability requirements by the grid cloud system and to process large number of computing requests.

The methodology will be as follows:

1. to identify the attributes of IaaS and describe the associated resources.
2. to build a grid cloud computing manager for executing high CPU jobs on the resources available on the grid clouds.
3. to present the architecture of our Agent-based Manager for the system with its modules: the Scheduler module to divide jobs to tasks, if possible, and distribute them over the available resources, and the Monitor module to monitor the resources and tasks. These modules are designed using Agents to increase interoperability, independently and scalability of the system.

4. finally to test this manager on real clouds (Google Compute Engine, Windows Azure) “selected from Table 1.1” to evaluate the AMGCS performance in terms of execution time and system throughput.

In order to use the aforementioned clouds we signed up for an account and purchase the computing services (Infrastructure-as-a-Service) to be used to execute our jobs. Google Compute Engine has a variety of scalable services to select from, its API will be integrated programmatically with our manager to be able to deal with the cloud and execute our tasks with high scalability level according to the QoS. The same with Azure Compute, we have purchased IaaS services and integrated its API with our manager to execute jobs on Azure infrastructure. Our manager has been implemented to manage and monitor the execution of jobs using the combined agents and APIs, tested with real high-computing jobs (scientific/mathematical analysis) to evaluate our manager and how the grid cloud computation perform tasks according to the specified QoS criteria.

Table 1.1: Selected list of clouds

Cloud Name	**URL**
Google AppEngine	http://developers.google.com/appengine/
Google Compute Engine	http://developers.google.com/compute/
Amazon EC2	http://aws.amazon.com/ec2/
AppScale	http://github.com/AppScale/appscale
Amazon S3	http://aws.amazon.com/s3/
Windows Azure	http://www.windowsazure.com/
Windows Azure Big Compute	http://www.windowsazure.com/en-us/solutions/big-compute/
Zimory	http://www.zimory.com/
Rackspace	http://www.rackspace.com/
Salesforce1	http://www.salesforce.com/
SpotCloud	http://www.spotcloud.com/

1.4 Thesis Organization

Thesis structure is organized as follows:

Chapter 1 introduces the research subject, provides overview about the problem statement and addresses the objectives and motivation of this research.

Chapter 2 presents a brief overview of the Grid and Cloud systems, their characteristics and the architectural models of these systems and their applications. It also presents an overview of the Software Agents, its definition and intention.

Chapter 3 presents a related work to this research and a literature review of cloud federation. It concludes these works and discusses them in the research.

Chapter 4 presents the architecture and design of Agent-based Manager for Grid Cloud System (AMGCS) in details, with explanations of system’s characteristics and services.

Chapter 5 discusses the implementation details of the AMGCS, explaining its detailed services and metadata structure. Multiple tests will be conducted and presented to clarify the system’s functions and performance.

Chapter 6 shows and discusses the experiments and tests performed on the AMGCS, evaluating its performance and throughput. It also compares the results with a single cloud system and shows the difference in optimization and performance.

Chapter 7 concludes this research ideas and points out the limitations, advantages and the future works that can be done to extend the scope of this research.

Chapter 2

Background

This chapter gives an overview about Grid and Cloud systems, discusses the characteristics, architectural designs and services of their environments. It also provides an overview about Software Agent, its characteristics, when and why to use it.

2.1 Grid Overview

The Grid is the collection of computer resources from multiple locations to reach a common goal. Grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. A Grid is a system that integrates and coordinates resources that are not subject to centralized control, i.e. live within different control domains, and it uses standard, open, general-purpose protocols and interfaces to address fundamental issues such as resource discovery and resource access. Grid is also a system that deliver nontrivial qualities of service, by allowing resources to be used in a coordinated fashion to deliver various QoS “response time, throughput, availability, and co-allocation of multiple resource types” to meet complex user demands.

Grid technology was initially developed to enable resource sharing within scientific collaborations, and then they are used in different large-scale collaborations. Sharing in Grid is not only in database or in files but also includes software and computational resources. Sharing resources in Grid is necessarily highly controlled, with resource providers and consumers defining clearly and carefully, just what is sharing, who is allowed to share, and the conditions under which sharing occurs. Grid systems build virtual supercomputers, which consist of different machines and networks.

Grid computing offers a solution to intensive-computing problems. The grid-computing paradigm is the field of computing science that aims to offer a seamless, integrated computational and collaborative environment [5]. Computational grid has been defined as *"a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities*" [6].

The idealized features and properties that are required by a Grid system to provide users with a seamless computing environment are characterized as follows [7, 8]:

- ***Large scale***: a grid must be able to deal with a number of resources ranging from few to thousands, avoids potential performance degradation as a grid size increases.
- ***Geographical distribution*:** grid's resources may be located at distant places.
- ***Heterogeneity*:** a grid hosts software and hardware resources that vary ranging from data, files, software components or programs to sensors, scientific instruments, display devices, computers, super-computers and networks.
- ***Resource sharing*:** resources in a grid belong to many different organizations that allow other organizations (i.e. users) to access them. Nonlocal resources can thus be used by applications, promoting efficiency and reducing costs.

- ***Multiple administrations*:** each organization may establish different security and administrative policies under which their owned resources can be accessed and used. As a result, the already challenging network security problem is complicated even more with the need of taking into account all different policies.

- ***Resource coordination***: resources in a grid must be coordinated in order to provide aggregated computing capabilities.

- ***Transparent access*:** a grid should be seen as a single virtual computer.

- ***Dependable access*:** a grid must assure the delivery of services under established QoS requirements. The need for dependable service is fundamental as users want to guarantee they will receive predictable, sustained and high levels of performance.

- ***Consistent access*:** a grid must be built with standard services, protocols and inter-faces thus hiding the heterogeneity of the resources while allowing its scalability. Otherwise, application development and pervasive use would not be possible.

- ***Pervasive access***: the grid must grant access to available resources by adapting to a dynamic environment in which resource failure is commonplace.

Using these properties and features, we can define a grid as geographically distributed hardware and software infrastructure composed of heterogeneous aggregated resources, owned and shared by multiple organizations coordinated to provide transparent, dependable and consistent computing support to a wide range of applications. These applications can perform distributed computing, high throughput computing, on-demand computing, data-intensive computing or collaborative computing [8].

2.1.1 Grid Architecture

The focus of a Grid architecture is on the interoperability and protocols among providers and users of resources to establish the sharing relationships. The required protocols are organized into layers, presented in figure 2.1, according to [12].

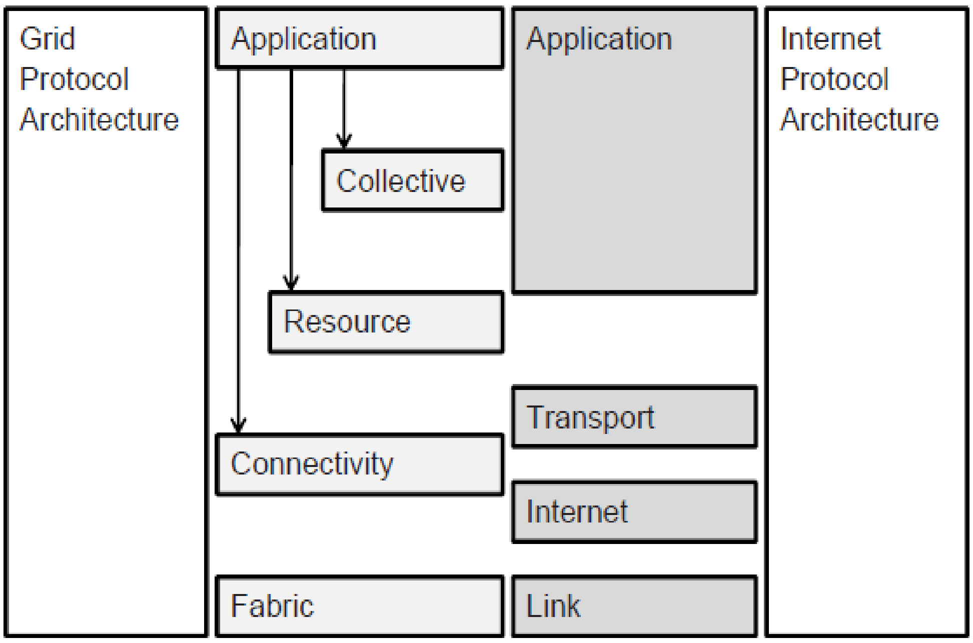


Figure 2.1: Grid architecture

The functionality of each layer is summarized as follows:

- **Fabric layer** comprises the physical resources that are shared within the Grid, i.e. computational, storage and network resources and software modules.
- **Connectivity layer** "*contains the core communication and authentication protocols required for a Grid-specific network transaction*" [12]. These protocols enable the data exchange between resources of the fabric layer.
- **Resource layer** uses the communication protocols from the connectivity layer to control negotiation, initiation and monitoring for the sharing of functions of individual resources, it comprises information and management protocols.

- The **Collective layer** is responsible for all global resource management and for interaction with collections of resources, and this layer's protocols implement sharing behaviors. Its functionalities include directory services, co-allocation, scheduling, monitoring and diagnostics services and data replication services.

- The **Application layer** involves the user Grid-enabled application that is deployed on the Grid, i.e. an application that is designed to run in parallel and use multiple processors of a Grid setting.

These layers of Grid Computing are interconnected and depend on each other; each layer uses the interfaces of the underlying layer. Together they create the Grid middleware and provide a comprehensive set of functionalities necessary for enabling reliable and efficient sharing of resources.

As a Grid is a collection of resources, i.e. compute, storage, communication and software; some of these resources may be used by all grid users while other resources may have restrictions. The following are the main components of a Grid system:

- **Computation Resources**

Computing is the most common resource provided by the grid machines' processors. The processors vary in speed, architecture, software platform, memory, storage, and connectivity. There are three main ways to utilize computation resources of a grid [9]. The first is to use it to run an existing application on an available machine on the grid rather than locally. The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors. The third is to run an application that needs to be executed many times on different machines in the grid.

- **Storage Resources**

The second most common resource used in a grid is data storage resources. A grid providing an integrated view of data storage is sometimes called a data grid [10]. Each machine on the grid usually provides some quantity of storage for grid use, even if temporary. Storage can be a memory attached to the processor or it can be a secondary storage, using hard disk drives or other permanent storage media to increase the capacity and reliability of data.

- **Communications Resources**

The rapid growth in communication capacity among machines makes grid-computing feasible, compared to the limited bandwidth available when distributed computing was first emerging. Therefore, another important resource of a grid is data communication capacity and communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of data to be processed, that may not reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid.

- **Special equipment and architectures**

Platforms on the grid often have different architectures, operating systems, capacities and equipment. Each of these represents a different kind of resource that the grid can use as criteria for assigning jobs to machines, while some software may only run on specific architectures. Such attributes must be considered when assigning jobs to resources in the grid, and the administrator of a grid may create a new artificial resource type that is used by schedulers to assign work according to policy rules.

2.1.2 Grid features

When a grid is deployed it must meets a set of requirements, and in order to match the grid-computing capabilities to these requirements, it is better to keep in mind common motivations for using grid computing [11]. Some of these motivations can be explained in terms of the grid as a high throughput computing system and high performance computing system. The main motivations for using Grid are:

- **Fault Tolerance and Reliability**

If a job submitted for execution at a specific node in the grid, the job allocates appropriate resources based on availability and the scheduling policy of the grid. Now if that node crashes for some reason, the grid makes provision for automatic resubmission of jobs to other available resources. To illustrate this concept, take as an example the Data Grids, which are grids for managing and sharing a large amount of distributed data. They serve multiple purposes and can be used to increase data transfer speed. Several copies of data are created in geographically distributed areas, whenever a user needs the data for computational purpose it can be accessed from the nearest machine hosting the data. Hence increase overall computational efficiency.

- **Balancing and Sharing Varied Resources**

Balancing and sharing resources provides the necessary resource management features. This aspect enables the grid to equally distribute tasks to the available resources. If the system in the grid is over-loaded, the scheduling algorithm can reschedule some tasks to other systems that are idle. In this way, the grid-scheduling algorithm transparently transfers the tasks to a less loaded system, making use of the underutilized resources.

- **Parallel Processing**

Some jobs can be broken into multiple tasks, each of which could be run on a different machine. Such jobs can be written to run as independent tasks and then the results from these tasks combined to produce the output. However, there might be constraints on the types of jobs that can be partitioned or a limitation on the number of tasks into which a job can be divided, to maximize the performance. If two of these tasks are running on the same set of data, then some locking mechanism similar to semaphores in operating systems must exist to guarantee data consistency. Therefore, constraints exist on the type of job to make it grid-enabled application.

2.1.3 Grid Applications

The different types of computing support offered by grids can be categorized according to the challenges they represent from the grid architecture point of view. The categorizations include the following:

– ***Distributed supercomputing support****:* Allows applications to use grids to couple computational resources in order to reduce the completion time of a job or to tackle problems that cannot be solved on a single resource.

– ***High-throughput computing support***: Allows applications to use grids to utilize unused processor cycles to work in loosely coupled or independent tasks.

– ***On-demand computing support****:* Allows applications to use grids in order to retrieve re-sources that cannot be cost-effectively or conveniently located locally.

– ***Data-intensive computing support****:* Allows applications to use grids to synthesize new information from distributed data repositories, digital libraries and databases.

2.2 Cloud Overview

There are many definitions for the term Cloud Computing from academics, analyst and industry practitioners. They differ in the description and definition of Cloud Computing from the perspective of the provider, end users, architectural aspects and other perspectives. From a scientific literature, a detailed definition of Cloud Computing from the Berkeley RAD Lab [13] is the following:

"*Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS). The datacenter hardware and software is what we will call a Cloud. When a Cloud is made available in a pay-as-you-go manner to the general public, we call it a Public Cloud; the service being sold is Utility Computing. We use the term Private Cloud to refer to internal datacenters of a business or other organization, not made available to the general public. Thus, Cloud Computing is the sum of SaaS and Utility Computing, but does not include Private Clouds. People can be users or providers of SaaS, or users or providers of Utility Computing*" [13].

The data center is the main Cloud component that contains the physical hardware resources for storage and computing, these two services together with software are offered in a pay-as-you-go manner. One characteristic of Cloud Computing is the integration and combination of hardware and system software with applications, that is integration of utility computing and SaaS.

Another definition by Foster et al. [14]: “large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet”. This definition shows two aspects, virtualization and scalability. The virtualized resources are provided through an Application Programming Interface (API) or a service. Resources, at the hardware level, can be added or removed according to the demand received through the interface, which itself is not changing to the user. This allows more flexibility and scalability on the physical layer of the Cloud without any impact on the interface to the end user. Figure 2.2 illustrates the Cloud Computing concept.

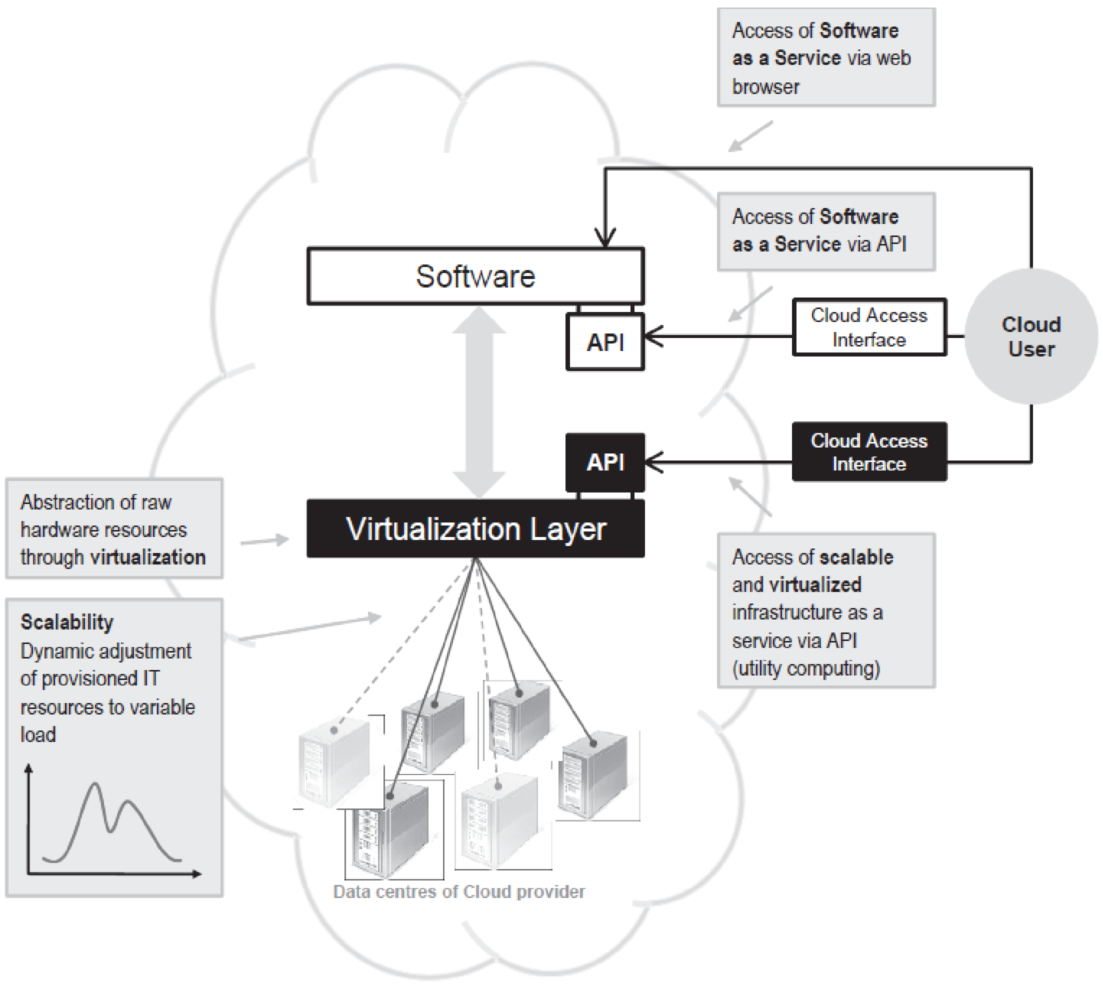

Access of **Software as a Service** via web browser
Access of **Software as a Service** via API
Software
API
Cloud Access Interface
Cloud User
API
Cloud Access Interface
Virtualization Layer
Abstraction of raw hardware resources through **virtualization**
Scalability Dynamic adjustment of provisioned IT resources to variable load
Access of **scalable** and **virtualized** infrastructure as a service via API (utility computing)
Data centres of Cloud provider

Figure 2.2: Cloud computing concept

In other words, the Cloud is a large group of interconnected machines extends beyond a single enterprise. Resources of the cloud such as storage, hardware, network and software are provided in a *X-as-a-Service* manner.

The most important features of Clouds are the virtualization and the dynamic scalability on demand, so a cloud system consists of a group of virtualized computers dynamically provisioned as one or more unified computing resource.

All Cloud services are offered through the Internet to users across multiple platforms, via a defined API or Web browser, depending on the user's usage. In cloud computing platforms, resources need to be dynamically reconfigured virtualization, consumers' requirements can vary over the time and amendments must be accommodated.

2.2.1 Cloud Architecture

In literature, there are a number of concepts for Cloud structures, these classifications may appear to be different from one another to varying extent, but finally they describe and classify a related structure and share a common denominator [15-17]. Most of these concepts do not provide a description that is a sufficiently generic for Cloud structure and its components, but the concept that is used to describe a generic structure and components of a cloud is the *three-layered* concept. Figure 2.3 shows the architecture of the Cloud.

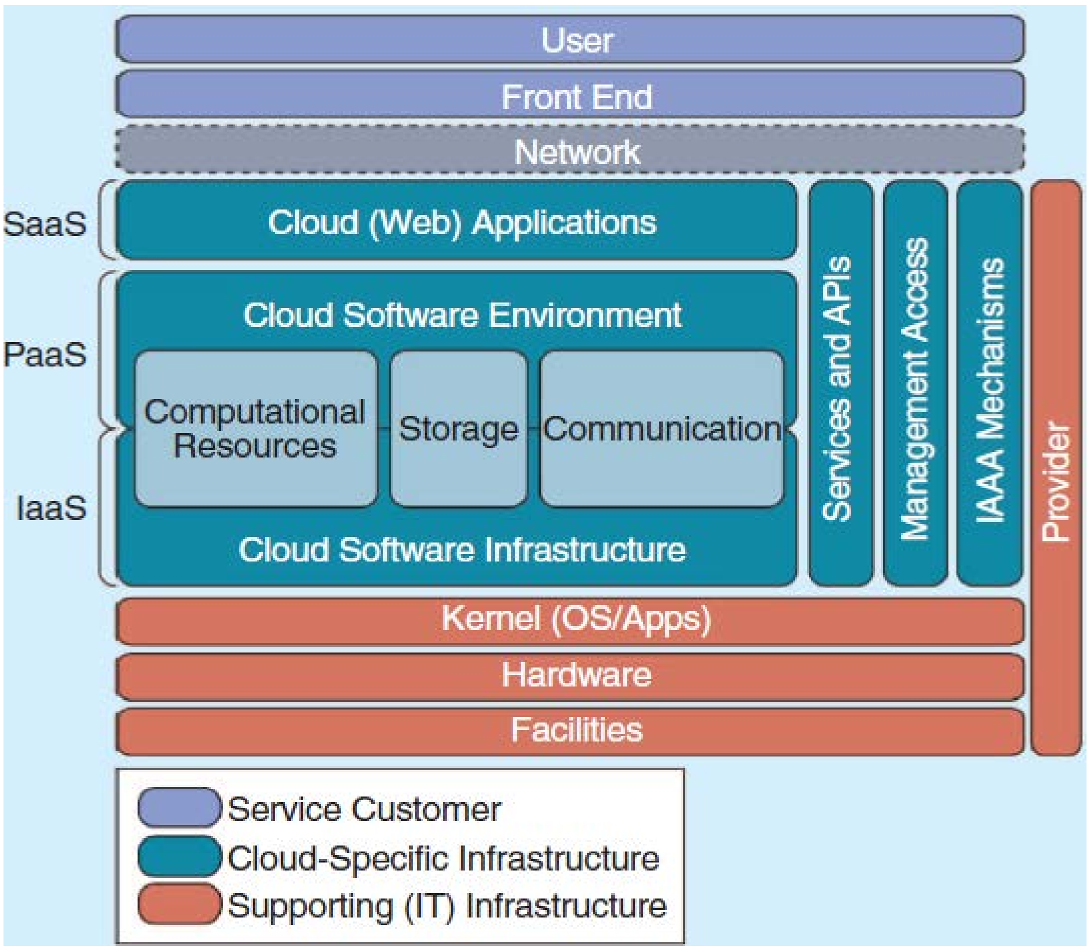


Figure 2.3: Cloud Architecture

Cloud Computing comprises different IT capabilities, namely infrastructure, platforms and software. These different "capabilities" may also be referred to as "layers", because *Infrastructure*, *Platform* and *Software* are built successively onto the forerunning level and are logically connected as different layers of the Cloud architecture. Therefore, the three architectural layers of Cloud Computing are: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [18].

The three layers are illustrated in Figure 2.4, where cloud services are related to the cloud infrastructure.

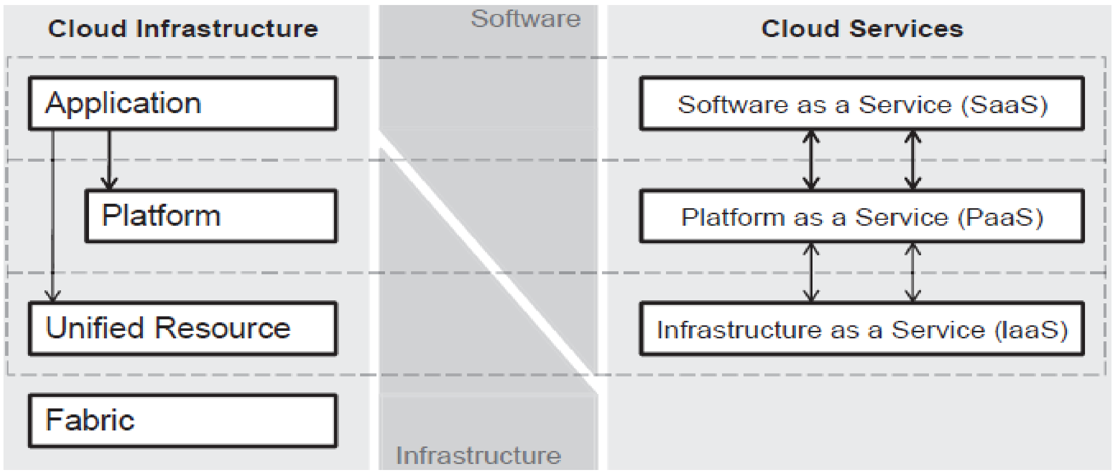


Figure 2.4: Cloud Layers

- Infrastructure as a Service (IaaS): offers storage, computing and network resources, accessed remotely with some benefits like pay per use, security, and reliability. Instead of selling raw hardware infrastructure, IaaS is offered as a virtualized infrastructure as a service. Examples are Google Compute Engine, Windows Azure Compute, Amazon Elastic Compute, Google Cloud Storage and Amazon Simple Storage.

- Platform as a Service (PaaS): is an abstraction layer between the virtualized infrastructure "IaaS" and the software applications "SaaS". Applications can be written and uploaded according to the specifications of a specific platform without worrying about underlying hardware infrastructure. Examples include Salesforce and Google App Engine, which allows applications to be run on Google's infrastructure.

- Software as a Service (SaaS): is the software which is owned, delivered and managed remotely by providers, offered in a pay-per-use manner. Because it is the actual software applications that are accessed and used, it is considered as the most visible layer of the Cloud for end-users. Examples are Google Apps (Gmail, Docs and Spreadsheets) and Salesforce.com.

2.2.2 Cloud Features:

The benefits of Cloud Computing are many; the major advantages of them could be summarized as follows: On-demand self-service, location-independent resource pooling, ubiquitous network access, transference of risk, lower costs, ease of utilization, quality of service, and reliability. For all different types of Cloud customers, the major opportunities known for X-as-a-Service offerings are offered by the Cloud. From the perspective of the user, the utility-based payment model is one of the benefits of Cloud Computing. There is no need for up-front infrastructure investment; software licenses, risk of unused but paid software licenses, hardware infrastructure, maintenance and staff.

Users of the Cloud services only use the volume or capacity of resources they actually need and pay only for the volume of resources they actually use, while they take advantage of the flexibility and scalability of the Cloud, as it enables easy and fast scaling of computing resources required on demand. Figure 2.5 shows different parties that can benefit from the Cloud.

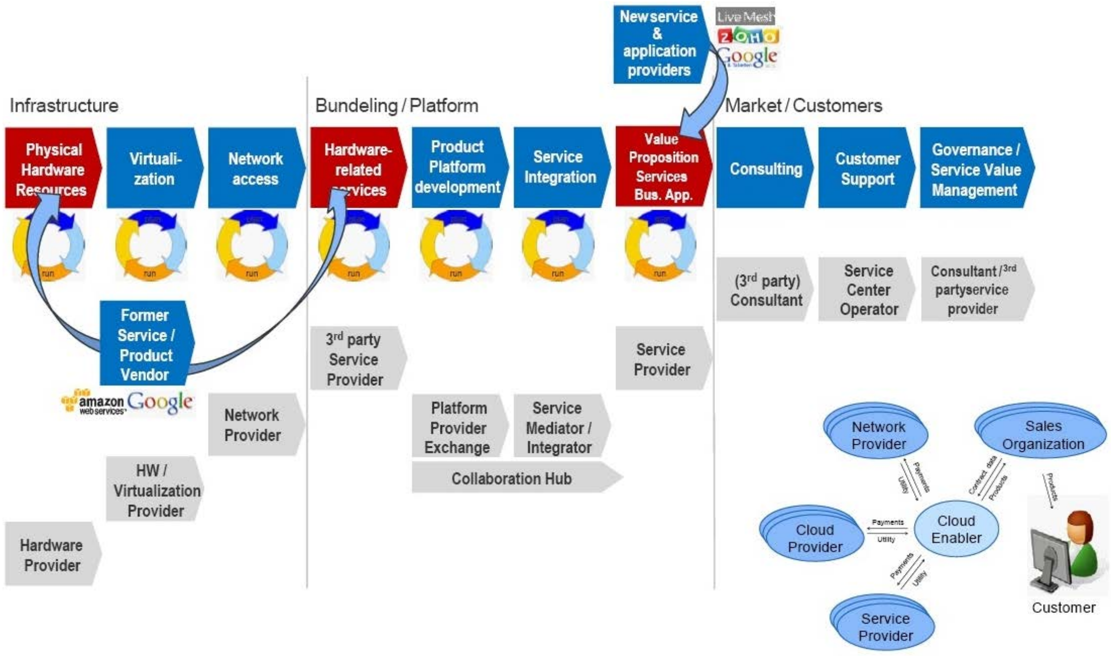


Figure 2.5: Cloud Parties

2.2.3 Cloud Models:

Cloud models are the description of the physical arrangement of the cloud infrastructure. Generally, clouds are classified according to the owner of the Cloud data centers. From cloud definitions aforementioned in section 2.2.1, Cloud Computing can be characterized as the obtaining of IT capabilities from external providers as a service. External data centers, e.g. those of Amazon, Azure or Google, are the foundation on the raw hardware level to deliver IT resources or capabilities as Cloud services. To differentiate between internal Cloud infrastructures "internal Clouds" and external providers of Cloud services "external Clouds", two main types of cloud models are commonly used: *Private* for internal Clouds and *Public* for external Clouds [13, 17].

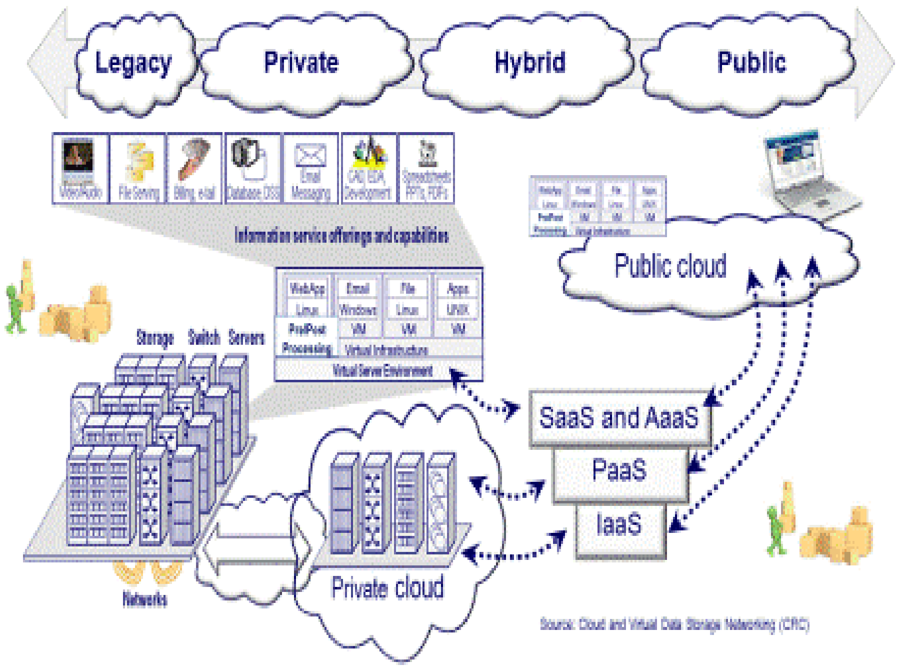
- **Public Cloud:** the most common type of cloud models, where data center hardware and software are run by third parties, e.g. Google and Azure, and expose their services to companies and consumers via the Internet. It is considered for web-based applications used by individuals and businesses where the application is a one-size-fits-all type of approach. It is not restricted to limited user base; it is made available in a pay-as-you-go manner to the public [13]. Examples of applications on public clouds include web-based emails, social media and online multimedia services.
- **Private Cloud:** refer to internal data centers, fully owned by a company who has control over the applications run on the infrastructure, where they run and the people using it [13]. This cloud is considered as dedicated for a given application or customer including being a part of the same physical location or in a remote location, and is customized to the exact needs of that application or customer. It relies on the virtualization of existing infrastructure for an organization, leading to increased utilization as described earlier. A key benefit here is the gain of all advantages of virtualization while retaining full control over the infrastructure [19].

As their names imply, public and private clouds differ by the degree with which the customer controls the configuration and who has access to the data. In a public cloud, there is little influence on the part of the customer, and the only interface is through the use of the web-based application. In a private cloud, the customer can control access, the platform type and even the hardware used; but the capital outlay is greater. Private clouds are also used when the customer is geographically diverse and wants to centralize their technology needs so that it is accessible from any of their locations [20].

Public and private clouds represent the two main cloud models, but there are other adopted models like Hybrid cloud and Community cloud:

- **Hybrid Cloud:** where Public and Private Clouds are combined, and applications could be delineated such that some exist on a public cloud and others exist on a “secure” dedicated private cloud. This way, companies can benefit from scalable resources offered by external Cloud providers while keeping specific applications or data inside the firewall.

- **Community Cloud:** an emerged model of hybrid cloud arrangement for businesses that are in some way related to each other, for example, different departments within a large corporation even if the actual work being performed by these different departments was independent of each other. It is an attempt to receive the cost benefits of a shared tenancy of a public cloud with the security and control provided by a private cloud, this led to multiple businesses with similar needs using a single private cloud infrastructure. The physical infrastructure of the cloud can be local or remote to the businesses that it supports, and its ownership can be the supported businesses or a third party [20]. Figure 2.6 shows the different models of the Cloud.



Legacy
Private
Hybrid
Public
Information service offerings and capabilities
Public cloud
Storage
Switch
Servers
SaaS and AaaS
PaaS
IaaS
Private cloud
Networks
Source: Cloud and Virtual Data Storage Networking (CRC)

Figure 2.6: Cloud Types

2.3 Gird Computing VS. Cloud Computing

The description of Grid Computing in section 2.1 and Cloud Computing in section 2.2 show that there are similarities between Grid and Cloud Computing. This has motivated many discussions in commercial and scientific literature asking if Grids and Clouds are the same, if Cloud is only a new hype of marketing, or if there are significant differences between Grid and Cloud Computing.

Cloud computing is a computing technology that uses the Internet and remote servers to maintain data and applications. It allows consumers to use applications without installation and access their personal files at any computer through Internet access [13]. It also allows for more efficient computing by centralizing memory, processing, storage and bandwidth.

Cloud computing emerges from grid computing and provides on-demand resource provisioning. Grid computing may or may not be in the cloud depending on the type of users who are using it. If the users are integrators and system administrators, they care about how things maintained in the cloud, they install and virtualize servers and applications. If the users are consumers, they do not care how things are run in the system.

Grid computing requires using software that can divide and farm out pieces of a program to several thousand computers. One problem with the grid is that if one piece of the software on a node fails, other pieces of the software on other nodes may fail if that component does not have a failover component on another node. Problems can arise if components rely on other pieces of software to accomplish the computing tasks. Large system images and hardware associated to operate them can cost a large capital and operating expenses.

Grid and Cloud computing are scalable, scalability is accomplished through the load balancing of application instances running individually on a range of operating systems and connected using Web services, also CPU and network bandwidth is allocated and de-allocated on demand. The system's storage capacity increased and decreased depending on the number of instances, users and the amount of data transferred at a given time. Table 2.1 compares between Grid and Cloud Computing [19-22].

Table 2.1: Comparison: Grid Computing and Cloud Computing

Topic	*Grid Computing*	*Cloud-Computing*
The problem	Computation over large data sets, or of parallelizable Compute intensive applications. Problem areas are often in pure research or in compute-intensive commercial	On-demand scalability for all applications, including research, development and business applications.
Main Target Market	First – Academia Second – certain industries.	Industry and academia.
The consumer	A member of a defined grid community, or a function within a large company	Open to anyone who can pay
User motivation	Low cost for large computations and processing of large amount of data.	- On demand scalability - Lower IT infrastructure costs (operations, energy, personnel)
The capability offered	Access to computers	Access to services, VMs and applications.
Unit of work	Grid computing application; a batch job, or a group of parallel batch jobs with a storage service	Either a Virtual Machine instance dedicated to the user where anything can be run, or using an online application with a Storage Service
Administration	Distributed, Virtual organizations.	Centralized.
Means of utilization	Allocation of multiple servers onto a single task or job.	Virtualization of servers; one server to concurrently compute several tasks.
Typical usage pattern	To execute jobs, i.e. execution of a program for a limited time.	To support long-running services
Abstraction level	Low abstraction, details exposed	High abstraction level.

2.4 Software Agents Overview

Software Agent is an autonomous component that interacts with its environment and with other agents on a user's behalf. It is kind of software abstraction, which provides a convenient and powerful way to describe a complex software entity, defined in terms of its behavior. Software Agent has the following capabilities, which make it a unique solution suitable for specific circumstances [23]:

- Persistence; code is not executed on demand but runs continuously and decides for itself when it should perform some activity.
- Autonomous; agents have capabilities of task selection, prioritization, goal-directed behavior, decision-making, operating as a standalone process and performing actions without human intervention.
- Communicative; agents are able to engage other components through some sort of communication and coordination; communicate with users, other software agents, other software processes or they may collaborate on a task.
- Reactivity; agents are able to perceive and respond to changes in its environment.

So an agent is a computer system that is capable of autonomous actions, that is, deciding and figuring out what needs to be done to satisfy its objectives. A multi-agent system consists of a number of agents; interact by cooperating, coordinating and negotiation with one another. When several agents work together and draw on the broad collection of their capabilities to achieve a common goal, this is called Cooperation. While Coordination is the process of achieving the state in which agents’ actions fit in well with each other. The Negotiation is a process by which a group of agents communicate with each other trying to come to a mutually acceptable agreement on some matter [24].

2.4.1 Characteristic of Software Agents

Characteristics of Software Agents are many, these characteristics work together to make agent-oriented systems more flexible, the following are some of them [25]:

- Autonomy: an agent is responsible for its own thread of control and can pursue its goal without being dependent on other agents.
- Adaptability: an agent's behavior may be changed after it has been deployed.
- Collaboration: agents communicating and working cooperatively with other agents to form multi-agent systems that are working together on some tasks.
- Knowledgeable: an agent is capable of reasoning about its knowledge and goals.
- Persistence: the infrastructure enables agents to retain knowledge and state over extended times, including robustness to face any potential run-time failures.
- Mobility: The ability to move from one context to another, either by moving the agent's code and starting the agent again or by serializing state and code, allowing an agent to continue execution in a new context with retaining its state.

In addition, software agent is defined in terms of its behavior, rather than being defined in terms of methods and attributes like other programming languages. Figure 2.7 shows the paradigm shifts of abstraction level of languages over the time.

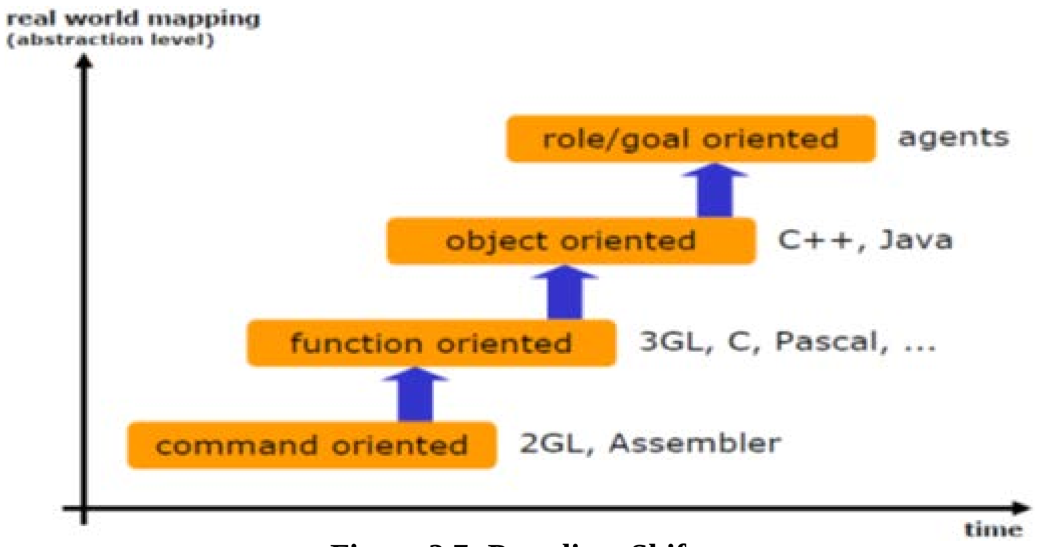


Figure 2.7: Paradigm Shifts

2.4.2 Types of Software Agents

- Personal agents: present some "personality" or "character", interact directly with a user, monitor and adapt to the user's activities, learning the user's style and preferences, and automate and simplify certain rote tasks.
- Mobile agents: to be sent to remote sites to aggregate and analyze data, collecting information or performing actions and then return with results. Such data-intensive analysis is better to be performed at the source of the data rather than transferring big raw data.
- Collaborative agents: communicating and interacting in groups, they represent organizations, users, and services. Multiple agents negotiate and share information through exchanging messages.

2.4.3 Why and when to use Software Agents

It is important to use high-level abstractions in building software that is complex to manage the complexity. An abstraction focuses on the essential and important properties of a problem and hides components that are incidental of that problem. Agents manage complexity by providing a new way of describing a complex process. Using agents make it easy to define a system in terms of agent-mediated processes.

Software agents are appropriate for use in a wide variety of applications. They are well suited for use in applications that involve communication between components or distributed computation, also for applications that reason about the objects or messages received over a network.

Multi-agent systems are also suited for applications that require distributed, concurrent processing capabilities. They can make it easier to build several kinds of complex systems. However, it must be remembered that software agents are appropriate for use to implement certain kinds of applications; but in other problem domains, other technologies could be more appropriate.

One of the benefits of the agent-based approach is that a complex processing function can be broken into several smaller, simpler ones. Since each individual agent can be crafted to be an expert in performing a particular task or solving a specific problem, systems could be built that show complex behaviors by using a group of relatively simple agents.

As known, essential characteristics of cloud computing include resource pooling and resource sharing. In agent-based cloud computing, the cooperation and coordination protocols of software agents are adopted to automate the activities of resource pooling and sharing in the clouds that are pooled to serve multiple cross-platform users [26].

Chapter 3

Related Work

There are projects focusing on cloud computing management with a variety of architectures and services. In this chapter, we study cloud resource management and mention some other systems that are related to this research, conclude these works and discuss them. We also present a literature review for Cloud Federation as a technique for integrating Cloud resources.

3.1 Cloud Resources Management

There are a number of management systems for cloud services, and some of these systems can be found as a locally installed management application with a GUI, command-line tools, extensions of a web browser or as online tools. They provide their own management interfaces, designed to specific needs without the ability to interact with other cloud deployments of the same system, particularized to work only with a specific cloud technology and not compatible with others. Some IaaS systems are replicating the same capabilities offered by public providers like Amazon AWS. These may include Nimbus, Eucalyptus, OpenStack and OpenNebula [27, 28].

We found some existing systems for managing grid of IaaS clouds, some of them are suitable for only services of one cloud [29], and others for multiple services from multiple providers, like '*Karlsruhe Open Application for cLoud Administration*'. It is a web based application for managing AWS compatible cloud services, allows for working seamlessly with a variety of services of various clouds like Google Storage, S3 and Walrus storage services. [30, 31].

There is also an open source, cross platform, cloud management system called *Scalr*; provides server management and auto scaling disaster recovery [32]. The manager has the ability to scale the virtual infrastructure according to the load based on RAM, disk, CPU, network or date.

Furthermore, there are open source initiatives like deltacloud [33], jcloud [34] and Libcloud [35], in addition to their limitation to a specific interface or programming language, they are mainly concerned with the management of public IaaS providers with basic support for some private IaaS systems. While they manage virtual instances, they do not concern about the underlying physical infrastructure. Other systems provided from academia offer a generic model for management to be adapted to any product. As their model was built with an exact technology, *REST* or *SOAP* web services, their interface is fixed. It is not easy to extend them to offer other kind of interfaces such as web pages or command-line tools.

If we look to other related works that use software agents in the management of the grid clouds, we can find some examples like [36]; they simulate a proposed framework based on agents to manage resources for service workflows, with a hierarchical architecture for separating decisions of resource management on service, workflow and cloud levels.

Others proposed an adaptive model for resource allocation for finding a proper data center according to the consumer location and the data center workload, also simulated using the agent-based testbed [37]. Another prototypal implementation found for an interface that is compliant with *Open Cloud Computing Interface* [38] to manage IaaS resources. This interface developed as an entryway to a standard FIPA multi agent system, offers services for IaaS management and resources negotiation [39]. Another group has presented an agent-based protocol for cloud service discovery. Taking advantage of an ontology description (semantic description of each resource), they developed a multi-agent system by introducing an ontology-based matching, using database for keeping track of historical data to make recommendations based on the prediction of the attribute value [40].

We also found that the number of works that involve the use of software agents in the process of managing grid clouds are limited, most of them either for resource negotiation / brokering or they are just a simulated ideas for resource allocation without implementation on real clouds [41, 42, 43]. We will study the works aforementioned, their functions of management and how they were designed. The motivation for choosing these systems is to study the resource management architectures, especially in the computational cloud system at IaaS level.

3.2 Cloud Federation

Cloud Computing offers three main service models; Infrastructure as a Service ‘IaaS’, Platform as a Service ‘PaaS’ and Software as a Service ‘SaaS’, in addition to others. IaaS services include Compute Clouds and Cloud Storage. PaaS provides platforms and execution environments, and SaaS provides only software. We can arrange these models as a stack as they relate to each other. The IaaS layer is the lowest level close to the underlying hardware. In this layer, we can distinguish two service types: computational and storage. Examples of clouds that provide infrastructure services are Google Compute Engine, Amazon EC2 and Azure Compute (Table 1.1). Next layer is the PaaS layer; examples include Amazon Elastic Beanstalk, Windows Azure and Google AppEngine. The third layer, SaaS e.g. Google Docs, is based on IaaS or PaaS.

In this research we are focusing on the IaaS layer as it provides the compute and storage services, we mainly use the computing services from different clouds using our proposed manager to execute jobs on these clouds. Grid clouds are similar to the concept of cloud federation, where services comprised from different clouds are aggregated together.

The term Federated Cloud has been used interchangeably with hybrid cloud in that both described a mixture of public and private clouds that are aligned with the needs of a given customer. Recently, however, the federated cloud term is being used to describe a different kind of cloud business model that is more aligned with a utility network model. The physical cloud resources are themselves being considered as a service, and cloud providers are offering their resources for other providers to expand the global cloud coverage offered to their customers without needing physical resources in every geographic locale.

In other words, a customer can obtain cloud services from a single provider, and that provider would obtain cloud resources in any geographical location via a local cloud provider, in a manner that is seamless to the customer and removes any latency concerns by being local to the point of use [20]. Consequently, the cloud become a federation of infrastructure providers or alternatively there will be a federation of clouds, making a collection of clouds that interoperate together, i.e. exchanging data and computing resources through defined interfaces. In cloud federation, each single cloud remains independent but can interoperate with other clouds in the federation through standardized interfaces. Figure 3.1 shows the layers of the cloud with different types of integrations (Horizontal and Vertical).

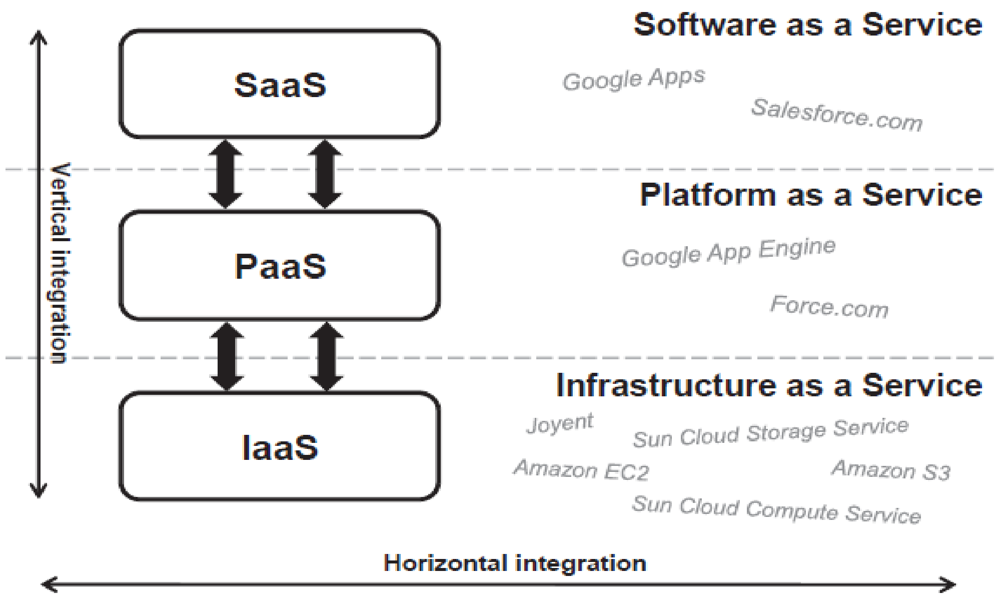


Figure 3.1: Integration types in the Cloud

There are two types of federation, Horizontal federation and Vertical federation. Horizontal federation expands the capacity of a cloud by integrating a new site and it takes place on one level of the Cloud Stack e.g., infrastructure level. Vertical federation allows the integration of new infrastructures to provide new capabilities by spanning multiple levels [44].

Presently, a Cloud Federation is still a theoretical concept, as there is no common standard for cloud interoperability. A new initiative is trying to develop a common standard is the *Open Cloud Computing Interface*, with a goal of a standardized API among clouds. This enables interoperability among diverse providers and enables a new business models and platforms like 1) "*Integrators*" for advanced management services spread over several Clouds, and 2) "*Aggregators*" for a single common interface to multiple Cloud providers. Interoperability and open standards between public and private clouds enable a high level of flexibility for uses, and users also would be able to partly outsource processes and data to clouds that are privacy-sensitive or less secure. The possibility of building federated clouds would enable specialization of single clouds as well as a broader choice for users [19].

One important point to be mentioned here is that Cloud Federation requires one provider to rent or sale computing resources to another provider. Those resources become a permanent or temporary extension of the buyer's cloud computing environment. Therefore, an agreement must be initiated between different cloud providers in order to make this integration "federation" valid.

The idea of managing grid cloud services, especially infrastructure services, emerges a new way of computing technology through grid cloud system. The related works that aggregate multiple clouds together are only simulated works, no real clouds are involved in their experiments. In contrast, the Agent-based Manager for Grid Cloud System is using real clouds. Furthermore, there is no need for an agreement between providers, as the Manager is integrating all APIs of these Clouds together and then managing resources and tasks using the proper API.

Chapter 4

Agent-based Manager for Grid Cloud System (AMGCS)

This chapter presents our work by providing the architecture of the Agent-based Manager for Grid Cloud System (AMGCS), with explanations on its functions and services. We introduce this chapter by providing a brief overview about Cloud management services, architectures and main phases in Cloud resources integration that AMGCS is based on in our study.

4.1 Grid Cloud Management Overview

Before we discuss our work in this thesis, we introduce this chapter by presenting an overview about some of the important aspects that affect the management in the grid cloud system. Architecture and services provided by the resource management system are affected by the type of the resources they manage. For computational grid cloud, the main resources that are being managed by the resource manager are the compute resources while in a data grid cloud the focus is to manage data distributed over cloud geographical locations. In this thesis, we only focus on a grid cloud that aggregates and manages the compute and storage resources.

Scheduling is an important component of the manager, which plays an important role in the overall performance of an application running on the grid cloud. Proper scheduling requires information about the status and availability of the resources in the grid clouds.

In a Grid Cloud management system, the following are frequently used terms:

- A ***resource*** is a term used to denote anything that can be scheduled and required to carry out an operation such as virtual machines, disk space, network and so forth.
- **A *job*** is any application that needs any kind of resources, i.e. bandwidth, compute, storage or any other resource to be run and to complete its tasks.
- The ***properties*** of a job are the parameters like memory or CPU requirements, priority, deadline, cost, reliability… etc.

4.1.1 Grid Cloud Management System Structures

A manager can be implemented in different structures, which determine the architecture of the resource management system and the scalability of the system. These structures are classified as ***centralized, hierarchical*** *or* ***decentralized***, shown in Figure 4.1.

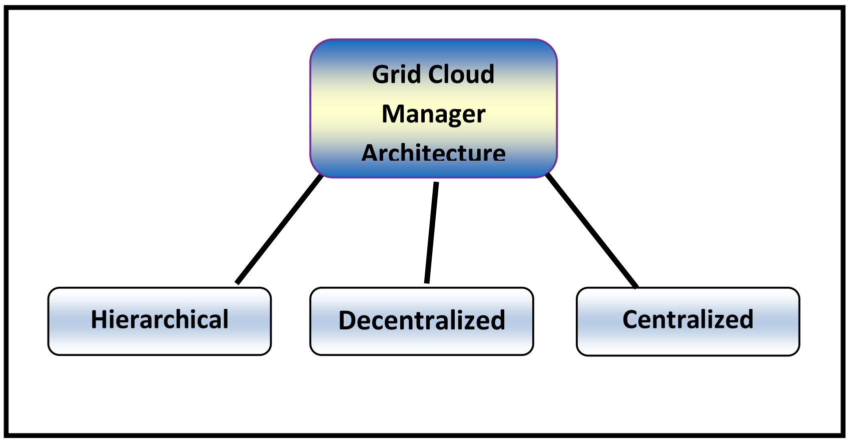


Figure 4.1: Categories of Grid Cloud Manager

- In a ***centralized*** model, jobs are submitted to a single manager that is responsible for scheduling them on the available resources. As all scheduling information is available at single place, the scheduling decisions are optimal but not very scalable in large systems. As the size increases, it would be difficult to keep all information about all resources states. Figure 4.2 shows a central manager architecture.

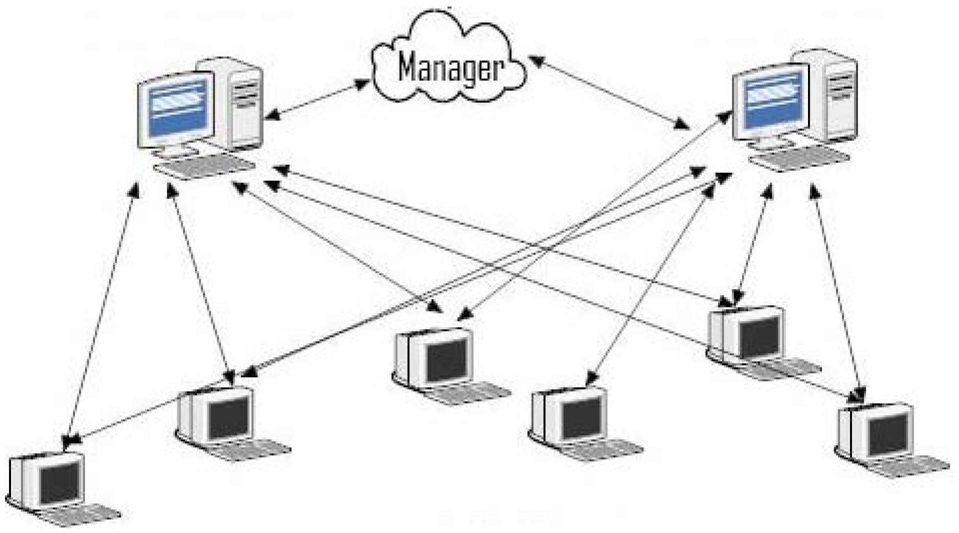


Figure 4.2: Centralized Manager Architecture

- In a ***decentralized*** model, there is no central manager as the managers are distributed on multiple locations. This approach is scalable and suits large systems. However, the managers should cooperate together in making scheduling decisions, and the generated schedule may not be the optimal one. It is perfect for peer-to-peer architectures and dynamic environments. Figure 4.3 shows a decentralized manager architecture.

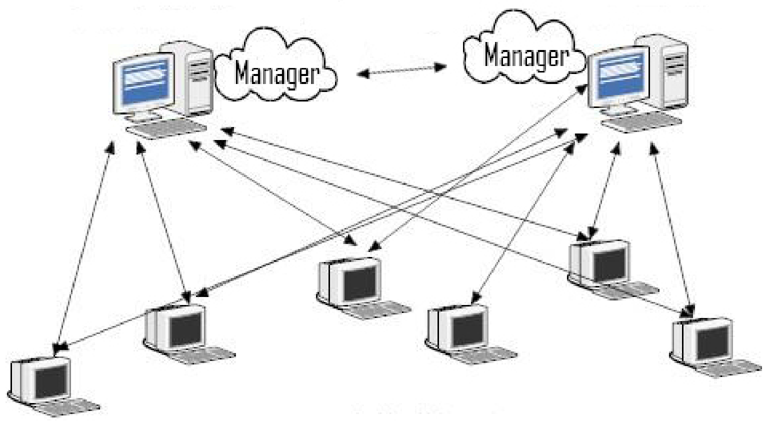


Figure 4.3: Decentralized Manager Architecture

- In a ***hierarchical*** model in Figure 4.4, the managers are organized into a hierarchy. High level resource entities are scheduled at higher levels and lower level. The smaller sub-entities are scheduled at lower levels of the manager hierarchy. This model is considered as a combination of the above two models.

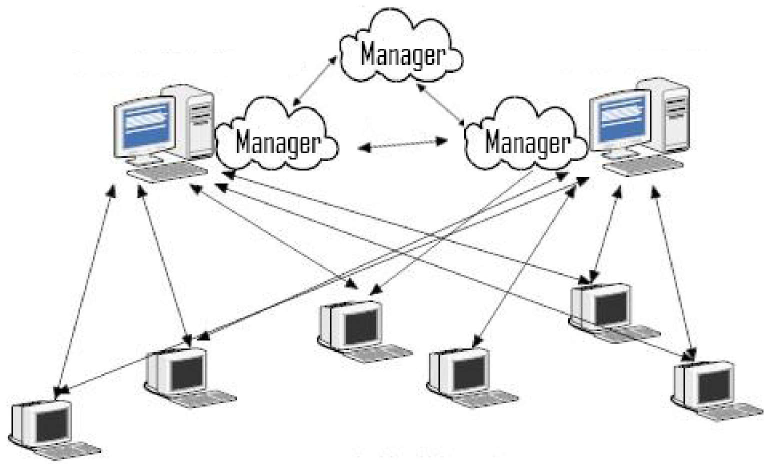


Figure 4.4: Hierarchical Manager Architecture

One way of classifying a manager is by the scope of its operation. A centralized manager schedules and manages all jobs submitted to the grid cloud, whereas a decentralized manager handles jobs submitted to a particular manager in the grid cloud. A centralized manager has a full knowledge and control on the resources and jobs. Hence, it can perform good scheduling, but easily become a single point of failure and a performance bottleneck. On the other hand, decentralized manager architecture scales well but with low optimal scheduling performance due to the multiplicity of managers.

Scheduling policies used by the grid cloud system can be classified into two major categories: *user-oriented scheduling* and *system-oriented scheduling*. User-oriented scheduling try to optimize the performance for an individual user by minimizing the response time for each job submitted by the user, whereas system-oriented scheduling often strives to maximize over system throughput, average response time, fairness or a combination of these. [46]. A decentralized manager uses a user-oriented policy, whereas a centralized manager performs system–oriented scheduling. The comparison between a centralized and decentralized manager is summarized in Table 4.1.

Table 4.1: Comparison between Centralized and Decentralized Manager

	Centralized Manager	***Decentralized Manager***
Scalability	Not scalable	Scalable
Fault tolerance	Single point of failure	More fault tolerance
Architecture	Client-Server architecture	Peer-to-Peer and dynamic environments
Information Storage	Keep information about all resources and jobs	Don’t keep information about all resource and jobs
Performance	System-oriented	User-oriented

4.1.2 Scheduling Phases

A grid cloud Scheduler is an important component in a management system. The user essentially interacts with the resource manager that hides the complexities of Grid Cloud computing. The grid cloud scheduler does not own the physical resources and therefore does not have control over them [47]; hence, the Scheduler must make best effort decision and submit the job to the resources selected.

In general, the scheduler function is to map jobs to the suitable resources in the grid cloud. The scheduler involves three main phases: *Resources Discovery, Resource Selection and Job Execution* (see Figure 4.5). Grid Cloud scheduling maintains a list of available resources and selects a best set of resources depending on users requirement and load balancing strategies. Then the scheduler dispatches the job to a selected virtual machine to execute it and finally it collects the results.

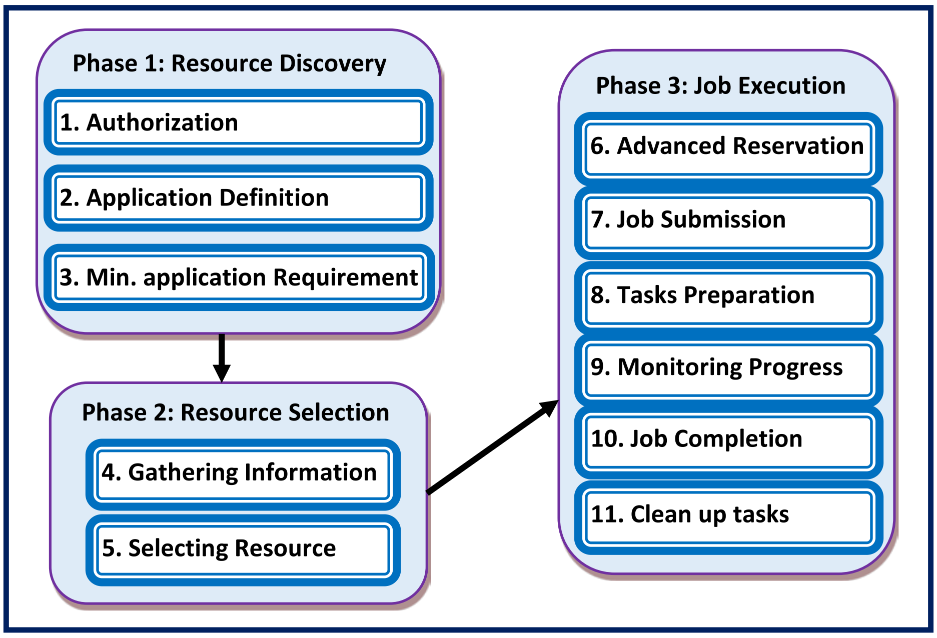


Figure 4.5: Scheduling Phases[1]

[1] This Figure is taken from Schopf, J., "Ten Actions When Grid Scheduling" [48].

Phase 1: Resource Discovery

Resource discovery is an important function of the resource management, used by the scheduling system to obtain information about the resources available. The goal of this task is to identify a list of authorized virtual machines that are available to a given user. Approaches to resource discovery can be classified as query-based or agent-based [49]. In the mostly used approach, query-based, the resource information store is queried for resource availability. In agent-based discovery, agents traverse the grid cloud system to gather information about resource availability. In this thesis, we use the agent-based approach but with alteration in the manner of discovering resources in the grid clouds.

Phase 2: Resource Selection

The scheduler selects resources to execute jobs depending on load balance algorithms. To select a resource, two steps must be done: gathering information about resources and making decision depending on expected starting time, the usage duration of resources, the CPU load on the resource… etc. [50]. The information about resources in the grid clouds is stored and maintained for scheduling uses.

Phase 3: Job Execution

When resources are chosen, the application can be submitted to the resources. Job submissions may be as easy as running a command or complicated as running series of scripts and may or may not require setups or staging. The simple acts of job submission can be complicated because of the lack of standards for job submission. The preparation stage may involve setup, stage, reservation or other required actions in order to prepare the resource to execute the job.

4.2 Agent-based Manager for Grid Cloud System (AMGCS) Overview

Resource management is important in the grid cloud system, its functions are to identify resource requirement, match and allocate resources, schedule and monitor them and to utilize them efficiently. Grid Cloud resource management focuses on the virtualization and coordinated use of heterogeneous and distributed resources. The current trend in Cloud systems is the adoption of the software agents for Grid Cloud architecture, as the software agent characteristics that we already presented in chapter 2 are compatible with cloud environments.

The compatibility of software agents' characteristics with Grid Cloud architecture allows us to design a manager for such architectures based on software agents. Using agents in managing grid clouds' resources enhances the interoperability, scalability and flexibility with high platform independence.

Different resources in a grid cloud are varying in operating systems, CPUs, VM images, memory… etc. This difference can lead to complex management for these resources. Software agent is well suited to address issues that arise from such a heterogeneous and remotely controlled but globally shared system. Dealing with changing requests and supporting autonomous resource mapping accentuate the need for cloud resource management systems, especially those systems that are capable of continuously managing the process of resource reservation by monitoring current service requests, amending future service requests, and autonomously adjusting schedules to accommodate dynamically changing resource demands [51].

Software agents are the most appropriate option for autonomously managing cloud resources in AMGCS, as users need to make decisions on selecting appropriate providers and negotiate with them to achieve “ideal” service contracts. Also providers need to make decisions on selecting appropriate requests to accept and execute depending on the resource availability, both current and future demands for services, and existing service obligations. Since agents are capable of making decisions when carrying out tasks on behalf of their users, and interacting with other agents through negotiation, cooperation, and coordination protocols, all of the above-mentioned requirements (challenges) motivates to adopt autonomous agents to allocate resources amid dynamically changing resource demands. Agent-based cloud computing is concerned with designing software agents for bolstering cloud service discovery, negotiation, and composition [52, 53].

In this thesis, we introduce an Agent-based Manager for Grid Cloud System (AMGCS) to group multiple clouds together (Figure 4.6) and run complex jobs that need high CPU by using the CPUs of the virtual machines in the grid clouds. The AMGCS aims to:

- Make full use of available computing power and built-in load balancing,
- Provide flexible access to virtual machines,
- Utilize cloud resources usage,
- Introduce an agent-based prototype for managing cloud resources and executing complex tasks using these resources,
- Be more scalable to more users and virtual machines.

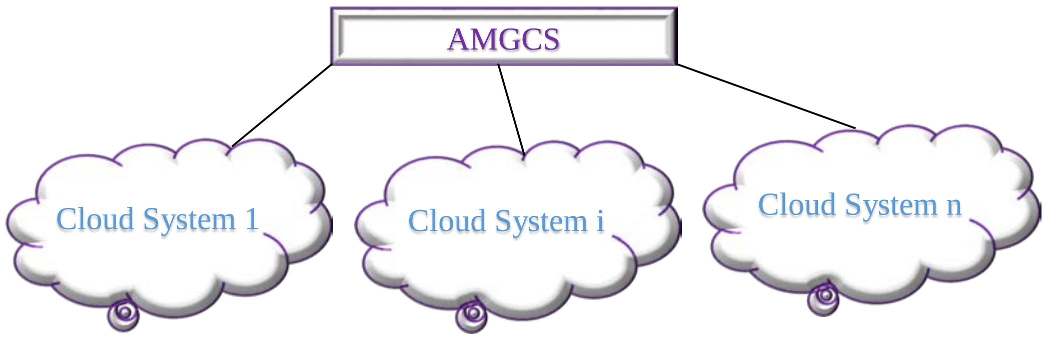


Figure 4.6: Managing integrated cloud systems

4.3 Grid Cloud concept

Grid cloud is built on the concept of Cloud Federation, which we mentioned in section 3.2. So a grid cloud is a way in which services characterized by interoperability features are aggregated from different clouds in one grid. It addresses the problems of vendor lock-in and provider integration, in addition to increasing the performance and the disaster-recovery process through techniques like co-location and geographic distribution. It may also enable further reduction of costs due to partial outsourcing to more cost-efficient regions. This concept satisfies some security requirements that might be necessary for some users, by using the fragmentation technique to execute part of the job on one cloud and the other part on another cloud, then combining results without allowing each cloud to know the actual job context.

As mentioned in section 3.2, the two types of federation are horizontal and vertical. Horizontal federation expands the capacity of a cloud by integrating a new site and it takes place on one level of the Cloud Stack, e.g. infrastructure level, illustrated in Figure 4.7a, showing integration link between different clouds on the same level. Vertical federation allows the integration of new infrastructures to provide new capabilities by spanning multiple levels [44], illustrated in Figure 4.7b.

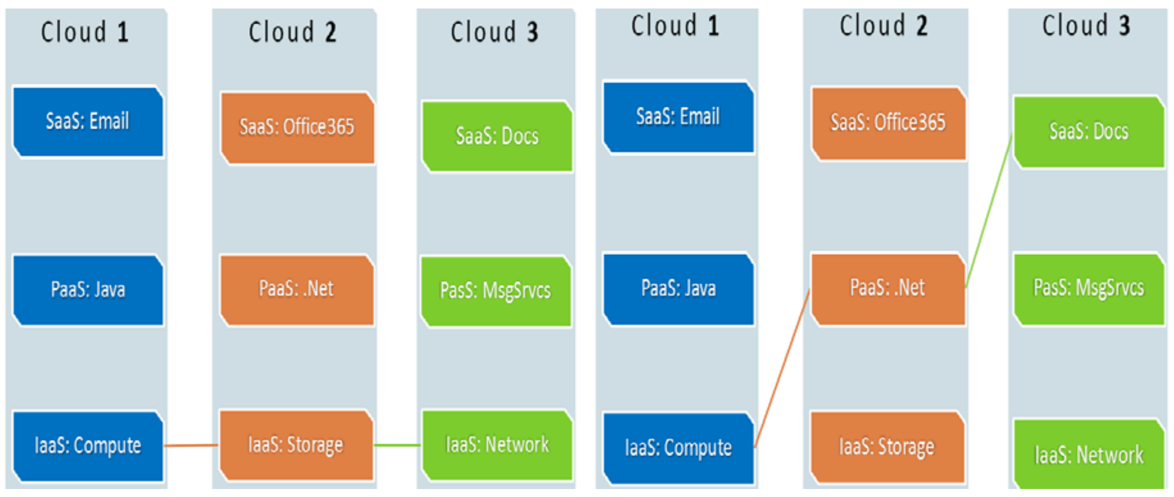


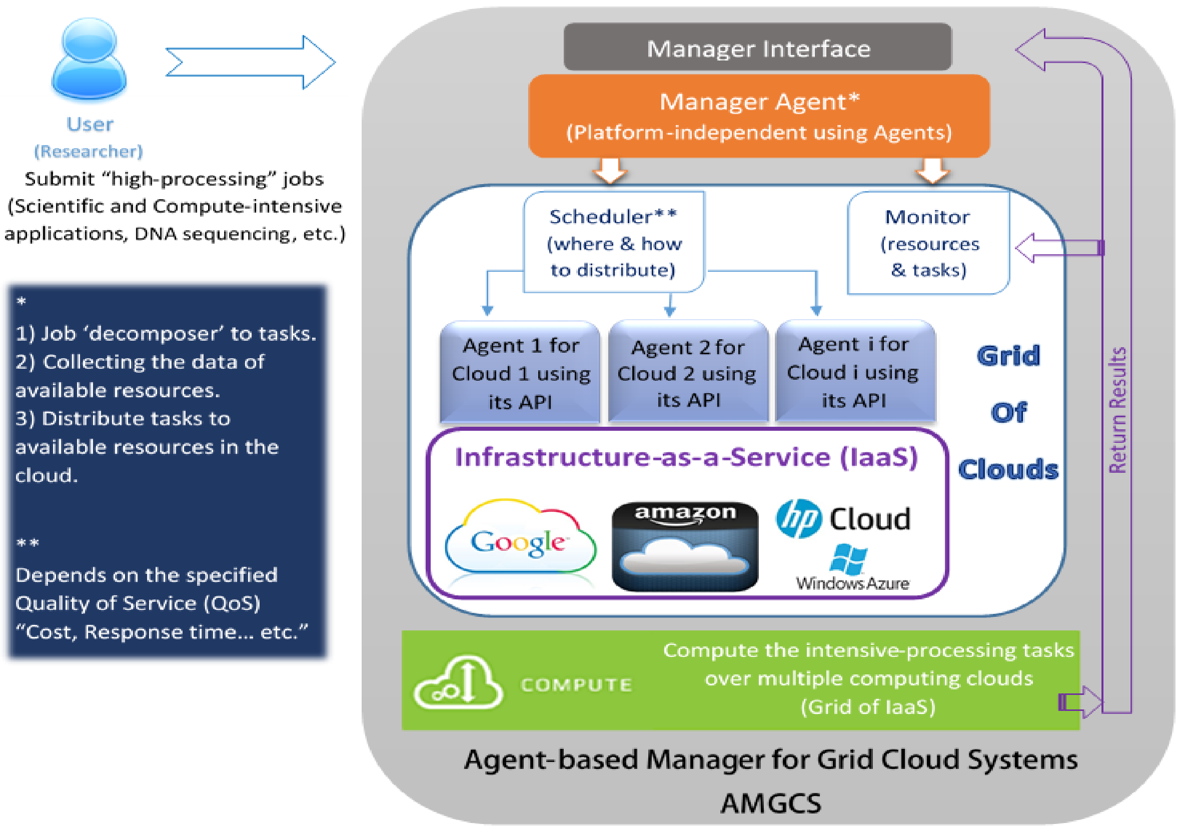
Figure 4.7: a) Horizontal Federation b) Vertical Federation

Applying this concept in our manager adds benefits like resource redundancy (parallel usage of similar services in different domains), resource relocation (data items, VM images or source code from domain to another) and combination of complementary services by combining different types to combined services [45].

Here we focus on the horizontal federation as it decreases the provider dependency and increases the availability (across multiple geographic regions). Therefore, if the QoS of executing jobs/tasks specifies the lower cost, for example, it can be executed on a cloud with the lowest cost or any other specific QoS required. Unlike Cloud Federation, AMGCS does not require an agreement between providers to integrate their services and resources, the manager itself combines the APIs of all grid clouds.

4.4 AMGCS Architecture

The general schematic of the Grid Cloud resource Manager that we designed in this thesis is illustrated in Figure 4.8.


User
(Researcher)
Submit "high-processing" jobs
(Scientific and Compute-intensive
applications, DNA sequencing, etc.)
*
1) Job 'decomposer' to tasks.
2) Collecting the data of available resources.
3) Distribute tasks to available resources in the cloud.
**
Depends on the specified Quality of Service (QoS) "Cost, Response time... etc."
Manager Interface
Manager Agent*
(Platform-independent using Agents)
Scheduler**
(where & how to distribute)
Monitor
(resources & tasks)
Agent 1 for Cloud 1 using its API
Agent 2 for Cloud 2 using its API
Agent i for Cloud i using its API
Grid Of Clouds
Infrastructure-as-a-Service (IaaS)
Google
amazon
hp Cloud
Windows Azure
Return Results
COMPUTE
Compute the intensive-processing tasks over multiple computing clouds (Grid of IaaS)
Agent-based Manager for Grid Cloud Systems
AMGCS
Figure 4.8: AMGCS structure

Because AMGCS provides different functions such as managing and monitoring resources and jobs, it consists of different agents, each of which is associated with a cloud API and managing resources on that cloud. The architecture model of AMGCS consists of two modules: a *Scheduler* and *Monitor* modules. The scheduler is responsible for managing resources available in the grid clouds, and allocates proper resources to jobs. The monitor module is monitoring jobs' executions and the resources reserved on the clouds for these jobs. These modules are interacting together in order to achieve their tasks properly, as illustrated in figure 4.9.

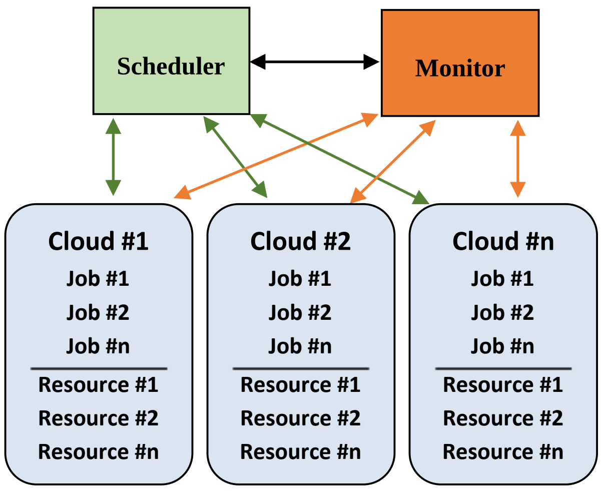


Figure 4.9: Modules of the AMGCS

These two modules are composed of sub-modules, each one associated with an agent, these sub-modules are *job scheduling*, *job decomposition*, *job manager*, *resources metadata*, each one with specific role but they are communicating together. The role of *job scheduling* is to schedule jobs according to available resources and user desires, *job decomposition* helps in decomposing job into tasks (based on job structure) to assign each task to a proper resource according to the information collected about all resources available in the grid clouds. *Job manager* monitors the execution of the jobs or tasks and the associated resources, finally the *resources metadata* sub-module collects and updates the metadata about available resources in all grid cloud systems.

4.4.1 Manager Services in AMGCS

AMGCS manages the virtual machines in the grid cloud system and is responsible for grouping multiple clouds together in the system; it schedules jobs depending on the metadata information then sends the job to the selected cloud to execute it on the associated virtual machine and returns the results. Moreover, it monitors execution of jobs and the utilization of available resources, also collects and updates metadata information about all available resources on the grid clouds, to create customized virtual machines to execute tasks and terminates these them after finishing their executions. Therefore, the manager here consists of number of services; each of which is associated with an agent and performs a specific function of the manager, by cooperating with each other to satisfy the manager responsibilities and goals. Figure 4.10 shows the algorithm used by the manager to achieve its services.

```
Input: Job J, User desires U, List of available resources R

Begin

Update metadata

Sort List R depending on U

If J can be decomposed, Decompose J into tasks JTs

If best-fit resource list BFR is not empty

Send J to BFR

Else send J to first-fit resource FFR

Execute J on selected virtual machine VM

Monitor J, R

Return results

End
```

Figure 4.10: AMGCS algorithm

Chapter 5

Implementation and Testing of AMGCS

This chapter discusses the implementation of the AMGCS. The techniques and tools used in building this manager with its integrated APIs. It also presents the architecture of each cloud system and an explanation on how to initialize the APIs used in the AMGCS manager.

5.1 AMGCS Implementation

The manager has been built using Java programming language to implement the agents that compose the manager itself. Each agent is implemented to perform specific tasks, decomposing jobs if it is possible, managing resources in different clouds and updating the metadata that contains the information about the available resources in all grid clouds. We have integrated our manager with multiple clouds APIs, these clouds are Google Compute Engine, Google Cloud Storage and Windows Azure Compute. Resources on these clouds are managed and controlled through the API functions of each cloud. In order to do that, we have had to understand their APIs and how to integrate them with the manager. Once we understand their functions, we can call them directly through our manager.

To be able to use these APIs, there must be an authorization and authentication processes that must be initiated before actual invoking of APIs functions and making requests. This initialization process must be done once, and then the manager will use these API functions to make any requests to manage our tasks and resources on these clouds, without the need of doing these steps again. We will go through these initialization steps briefly in each cloud we integrated with our manager.

5.2 AMGCS integration with Google Compute Engine

The API of the Google Compute Engine (GCE) has been integrated with our manager to directly call and request any details about the available resources and managing them. GCE first needs to authenticate the machine before accepting any request; this is done through the OAuth 2.0 protocol, which provides clients a method for accessing resources on behalf of a resource owner, like different clients or end-users. The OAuth 2.0 authorization framework enables a third-party application to obtain an access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf [54].

We have integrated the Google Compute Engine API with our Manager to get their latest machine types available, manage our resources on the GCE and to store and retrieve these data in the manager’s metadata. We used a cloud database to store this metadata, which is Google Cloud Storage, (section 5.5), to guarantee the compatibility and independency of any platform.

Figure 5.1 shows the architecture of Google Compute Engine and the different ways of accessing the cloud, either through the Command Line Interface (CLI), User Interface (UI) or the API code library using different programming languages including Java.

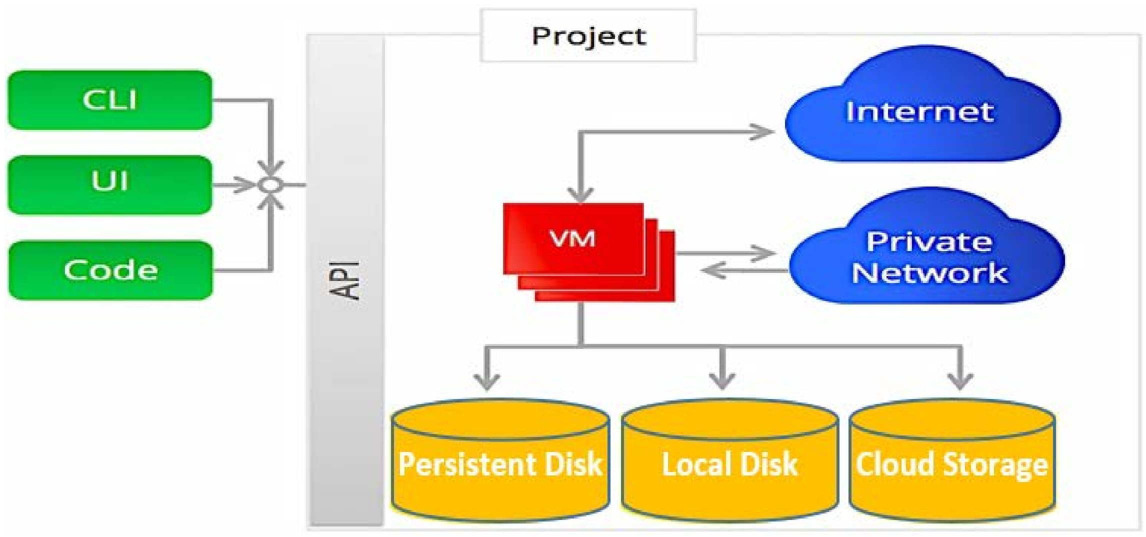


Figure 5.1: Google Compute Engine architecture

In this thesis we used the Java library for the API code to be integrated with the manager which will make requests through this API to manage the related resources and jobs. To use this API within our manager, we would need a key from Google Compute Engine that must be associated with our manager in order to be authorized to perform any requests, this key is called the "client_secrets" and can be downloaded in a JSON format from Google Developers Console, here is an example client_secrets.json file:

```
 {
  "installed": {
    "client_id": "837647042410-75ifg...usercontent.com",
    "client_secret":"asdlkfjaskd",
    "redirect_uris": ["http://localhost", "urn:ietf:wg:oauth:2.0:oob"],
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://accounts.google.com/o/oauth2/token"
  }
}
```

The manger will take advantage of OAuth 2.0 to authenticate the RESTful API to create and delete virtual machine instances, disks, and other resources, and to seamlessly integrate with other Google Cloud services such as Google Cloud Storage to store metadata that will be accessed by the manager itself to manage and schedule tasks according to this updated metadata information. Figure 5.2 shows the flow of authenticating APIs calls.

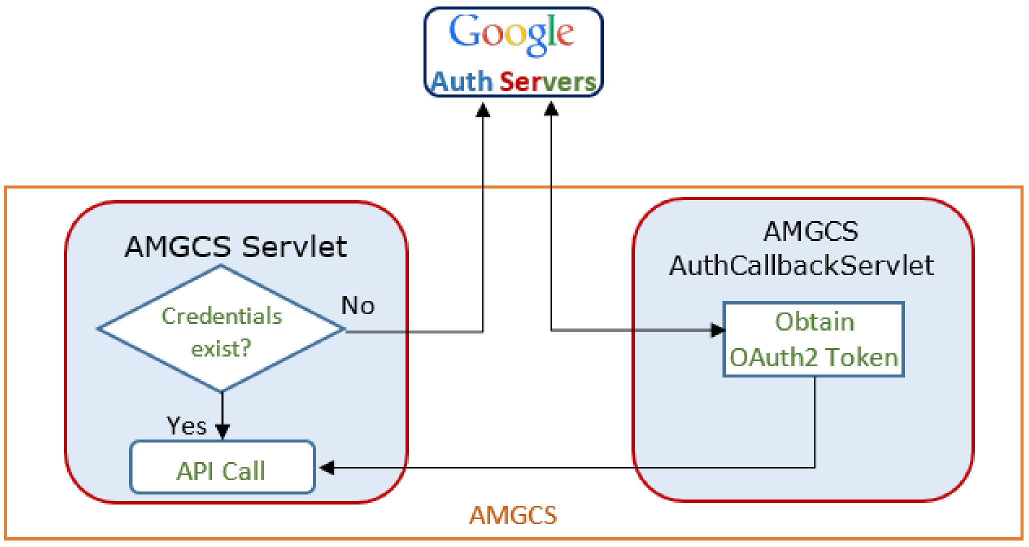


Figure 5.2: Authenticated API calls sample flow

The servlet is a Java class to extend the server capabilities to respond to any requests types and to extend the applications hosted by web servers. There are several machine types available from Google Compute Engine, categorized as micro, standard, high CPU and high memory machine types; Table 5.1 shows some of these machine types. AMGCS can select the high CPU machine types to be used for tasks that require more virtual cores relative to memory. Google Compute Engine uses GCEU (Google Compute Engine Unit) as a unit of CPU capacity describing the compute power, the minimum power of one logical core on the Sandy Bridge platform is 2.75 GCEUs.

Table 5.1: Selected list of machine types on GCE

Configuration	*Virtual Cores*	*Memory*
Micro - Small	*Shared*	*0.60 – 1.7 GB*
Standard	*1*	*3.75 GB*
	2	7.50 – 13 GB
High Memory,	4	15 – 26 GB
High CPU	8	30 – 52 GB
	16	60 – 104 GB

These machine types, in addition to many others, are included in our metadata database, so the scheduler will choose the proper one to execute the received job, according to the required QoS and whether the cost or the response time is the most critical factor for user desires. After the machine type has been selected, the API call will send a request, after being authenticated, to initiate a new instance with the specified configuration. Afterward, the manager calls the API function to start running the specified instance (virtual machine) on the Infrastructure after specifying the machine properties and configurations, using the *instances().insert* function. These instances can run Linux server from many images available; provided by Google or customized images of other systems, as we need. Finally, the selected jobs will be executed on this instance and others on other instances, results returned to the manager then to the user. The integration of the GCE API with our manager is illustrated in figure 5.3.

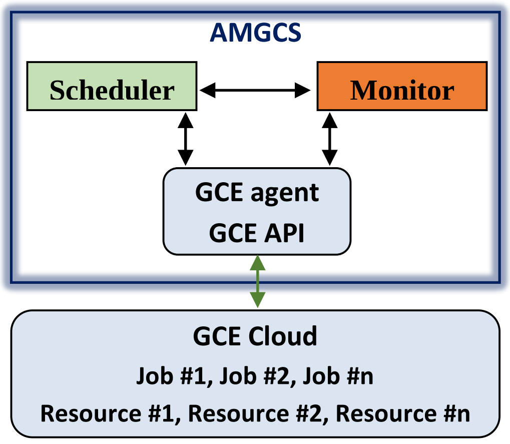


Figure 5.3: GCE integration with AMGCS

5.3 AMGCS integration with Windows Azure Compute

Windows Azure IaaS supports Microsoft operating systems and non-Microsoft ones. The VM image gallery in Azure Compute includes latest releases of Windows Server, SharePoint, SQL Server, BizTalk Server, and many non-Microsoft workload like Ubuntu, SUSE Linux, openSUSE, OpenLogic, etc. Integrating Azure Compute with AMGCS gives the power of handling yet more requests for high computing, with a built-in capability of Load Balancer, it also monitors VMs and restarts any that fail. *Roles* are the core of Windows Azure Compute; a role instance is a set of code, configuration, and local data, jobs are run in these instances (VMs) and, if required, data will be stored in the storage. AMGCS calls the associated API requests to create new instances, which are *Add Role* then *Start Roles* requests. The following schematic illustrates Windows Azure Compute (WAC).

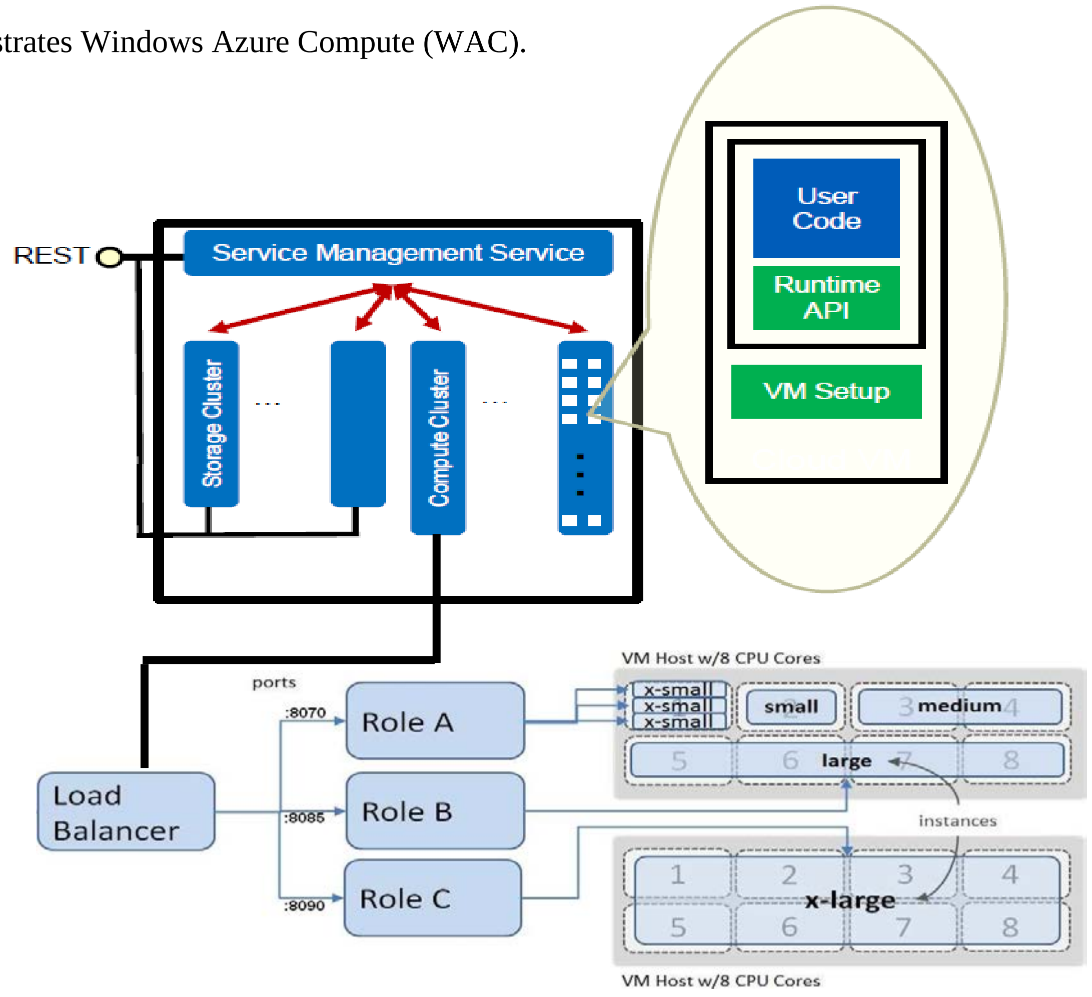


Figure 5.4: Windows Azure Compute architecture

Windows Azure Compute has many machine types ranging from extra small to extra-large machines, and also AMGCS will select from this wide range of machine types the proper machine type with proper configurations that suits the job requirements, Table 5.2 shows some of these machine types.

Table 5.2: Selected list of machine types on Azure Compute

Configuration	*Virtual Cores*	*Memory*
Extra small	Shared core	768 MB
Small	1	1.75 GB
Medium	2	3.5 – 14 GB
Large	4	7 – 28 GB
Extra large	8	14 – 56 GB

The integration of the GCE API with the manager's agents is illustrated in figure 5.5.

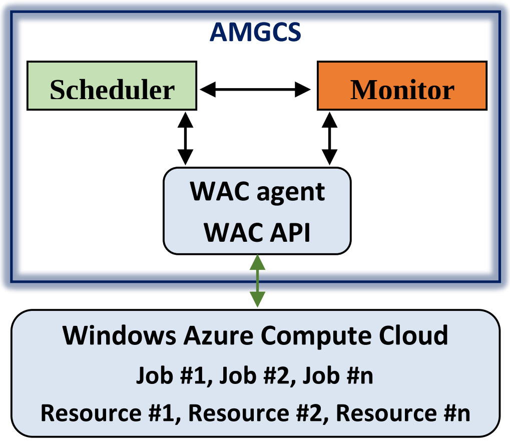

AMGCS
Scheduler
Monitor
WAC agent
WAC API
Windows Azure Compute Cloud
Job #1, Job #2, Job #n
Resource #1, Resource #2, Resource #n

Figure 5.5: Windows Azure Compute integration with AMGCS

In order to use this API, we would need the following:

1. A subscription Id: which uniquely identifies our subscription; we get this id from the Windows Azure portal.
2. A management Certificate: which is required to authenticate API calls, it must be associated with our subscription by uploading the certificate to the portal.

The API is a REST based API and so the endpoints are accessible over HTTP. In our code, we create an endpoint specific to a particular kind of operation that we want to perform, and then we create an HTTP request for that endpoint. To authenticate the request, we attach the management certificate with that request. The following are the steps of using Management Certificate:

<u>Step 1: Creating a Keystore:</u> We have to create a *Keystore* by using a tool called *Keytool*. The following command is used to create a *Keystore*:

```
keytool -genkeypair -alias mydomain -keyalg RSA -keystore WindowsAzureKeyStore.jks
-keysize 2048 -storepass "osama123"
```

We created a *Keystore* called "*WindowsAzureKeyStore.jks*" and set access password, the file created in "C:\Program Files\Java\jre7\bin" folder.

<u>Step 2: Exporting Management Certificate</u>: to export a certificate from this *Keystore* that we just created, also using *Keytool*, the following command was used:

```
keytool -v -export -file D:\WindowsAzureSMAPI.cer -keystore WindowsAzureKeyStore.jks
-alias mydomain
```

This will create a file called "*WindowsAzureSMAPI.cer*" in the "D:\" folder.

<u>Step 3: Uploading Certificate</u>: we login into the Windows Azure Portal and upload the file named "*WindowsAzureSMAPI.cer*" under the "Management Certificates" tab.

In the code, AMGCS needs this management certificate in order to make its requests authenticated. The management certificate is in the *Keystore* so we open it by the full path of the *Keystore* we have just created with password to get the *SSLSocketFactory*.

We have integrated the Windows Azure Compute API with our Manager to get the available Virtual Machine types, manage our resources on the Windows Azure and to send and receive messages between the Manager and the tasks being executed on the cloud, through "Windows Azure Service Bus, and Messaging Queue".

5.4 Using the integrated APIs through the manager in execution time

For Windows Azure Compute, our manager will execute the HTTP requests; this is done through the following scenario:

1) Creating a URL: based on the operation that we want to perform. In our case, our subscription id is “4bbac197-9348-4d19-8898-0e4baa009639”, and for example we want to perform operation “*List Locations*”, the URL would be: https://management.core.windows.net/4bbac197-9348-4d19-8898-0e4baa009639/locations

2) Creating HttpsURLConnection object: by using this URL, we create an instance of HttpsUrlConnection object and set the SSLSocketFactory.

3) Providing other necessary information: like required request headers, request method, content type… etc. then execute the request. For example, if we want to perform the operation “*List Locations*”, we need to perform a “*get*” request.

For Google Compute Engine, the AMGCS manager specifies an action also by using an HTTP verb such as GET, POST, PUT or DELETE. It specifies the resource by a globally unique URI in the following form:
https://www.googleapis.com/compute/v1/{resourcePath}?{parameters}

The returned data will be in the JSON (JavaScript Object Notation) format.

The metadata stored by the manager will be saved in the Google Cloud Storage, using the same account used for the GCE, the following section describes the metadata and how it was built on the Google Cloud Storage.

5.5 Monitoring tasks and resources

As we mentioned earlier, the manager has an agent for monitoring tasks and resources, this agent is specific to a particular cloud system, so we have a separate agents for Google Compute Engine and Windows Azure Compute. Each one of these clouds has its own API functions to get the status of the resources, and how tasks are being executed on these resources. So the manager is using these API functions or requests to monitor and get the status updates periodically. Figure 5.6 and 5.7 shows the monitoring charts for GCE and WAC respectively.

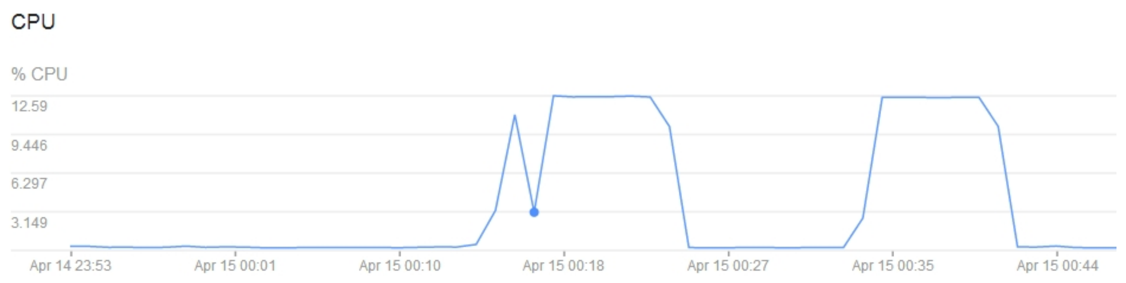
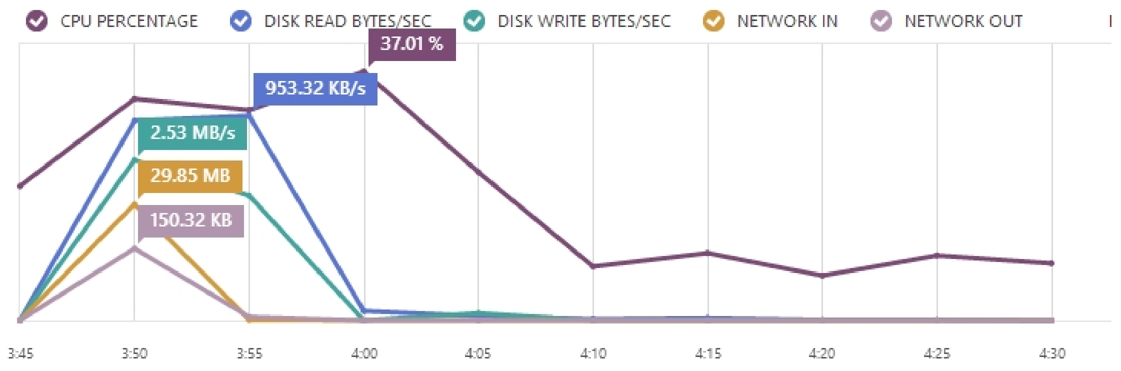


Figure 5.6: Monitoring GCE resources



	NAME	SOURCE	MIN	MAX	AVG	TOTAL
✓	CPU Percentage	osama	6.65 %	37.01 %	18.5 %	---
✓	Disk Read Bytes/sec	osama	54.97 B/s	953.32 KB/s	212.12 KB/s	---
✓	Disk Write Bytes/sec	osama	0 B/s	2.53 MB/s	507.82 KB/s	---
✓	Network In	osama	0 B	29.85 MB	3 MB	29.99 MB
✓	Network Out	osama	0 B	150.32 KB	15.77 KB	157.67 KB

Figure 5.7: Monitoring WAC resources

5.6 AMGCS Metadata

We have used the Google Cloud Storage (GCS) to store the AMGCS manager's metadata. This storage service is provided by Google, with features like object versioning, parallel uploads and CRC-based integrity checking to maintain the robustness of our sophisticated manager. We can access its API using XML, JSON or using the libraries for several popular programming languages including Java [55]. We used this storage service to guarantee the platform independency and the proper integration with agents associated with different clouds. The metadata include the following details from each cloud system in the grid cloud:

Name: name of the resource or virtual machine.

Description: description about the resource.

ID: The unique ID of the resource.

CPUs: Number of CPUs in the virtual machine

ImageSpace: The size of the server image in Gigabyte.

Kind: The category of the virtual machine, e.g. *high memory*, *high CPU*, or *standard*.

Disks: The maximum number of disks can be associated to a specific virtual machine.

DisksSize: The size of the disks associated to a specific virtual machine

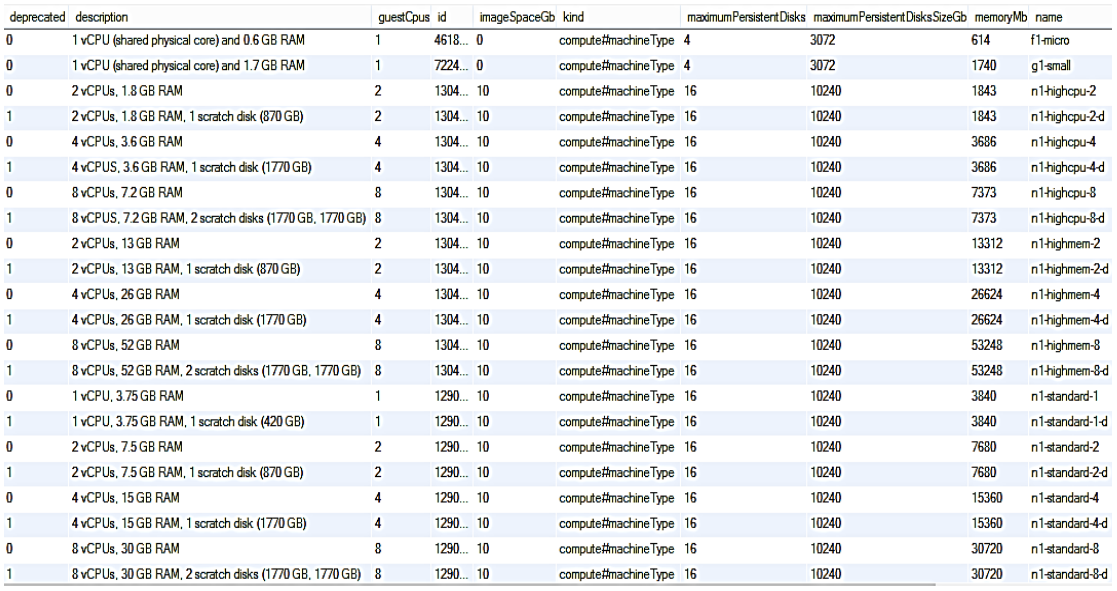
Memory: The size of memory in Megabyte.

Location: The location of the server, e.g. *Central US*, *West Europe*, *East Asia*… etc.

ServerType: The type of the server, e.g. *Windows*, *Linux*, *SQL*, *Oracle*…etc.

ServerImage: The image of the server, which contains the boot loader, an operating system and a root file system that is necessary for starting an instance, e.g. *debian-7*, *centos-6, rhel-6, sles-11*, *Windows Server 2012 R2 Datacenter*, *SQL Server 2012 SP1 Enterprise*, *OpenSUSE 13.1*, *Ubuntu Server 14.04 LTS*… etc.

Figure 5.8 is a snapshot of the current metadata collected about available resources.



deprecated	description	guestCpus	id	imageSpaceGb	kind	maximumPersistentDisks	maximumPersistentDisksSizeGb	memoryMb	name
0	1 vCPU (shared physical core) and 0.6 GB RAM	1	4618...	0	compute#machineType	4	3072	614	f1-micro
0	1 vCPU (shared physical core) and 1.7 GB RAM	1	7224...	0	compute#machineType	4	3072	1740	g1-small
0	2 vCPUs, 1.8 GB RAM	2	1304...	10	compute#machineType	16	10240	1843	n1-highcpu-2
1	2 vCPUs, 1.8 GB RAM, 1 scratch disk (870 GB)	2	1304...	10	compute#machineType	16	10240	1843	n1-highcpu-2-d
0	4 vCPUs, 3.6 GB RAM	4	1304...	10	compute#machineType	16	10240	3686	n1-highcpu-4
1	4 vCPUS, 3.6 GB RAM, 1 scratch disk (1770 GB)	4	1304...	10	compute#machineType	16	10240	3686	n1-highcpu-4-d
0	8 vCPUs, 7.2 GB RAM	8	1304...	10	compute#machineType	16	10240	7373	n1-highcpu-8
1	8 vCPUS, 7.2 GB RAM, 2 scratch disks (1770 GB, 1770 GB)	8	1304...	10	compute#machineType	16	10240	7373	n1-highcpu-8-d
0	2 vCPUs, 13 GB RAM	2	1304...	10	compute#machineType	16	10240	13312	n1-highmem-2
1	2 vCPUs, 13 GB RAM, 1 scratch disk (870 GB)	2	1304...	10	compute#machineType	16	10240	13312	n1-highmem-2-d
0	4 vCPUs, 26 GB RAM	4	1304...	10	compute#machineType	16	10240	26624	n1-highmem-4
1	4 vCPUs, 26 GB RAM, 1 scratch disk (1770 GB)	4	1304...	10	compute#machineType	16	10240	26624	n1-highmem-4-d
0	8 vCPUs, 52 GB RAM	8	1304...	10	compute#machineType	16	10240	53248	n1-highmem-8
1	8 vCPUs, 52 GB RAM, 2 scratch disks (1770 GB, 1770 GB)	8	1304...	10	compute#machineType	16	10240	53248	n1-highmem-8-d
0	1 vCPU, 3.75 GB RAM	1	1290...	10	compute#machineType	16	10240	3840	n1-standard-1
1	1 vCPU, 3.75 GB RAM, 1 scratch disk (420 GB)	1	1290...	10	compute#machineType	16	10240	3840	n1-standard-1-d
0	2 vCPUs, 7.5 GB RAM	2	1290...	10	compute#machineType	16	10240	7680	n1-standard-2
1	2 vCPUs, 7.5 GB RAM, 1 scratch disk (870 GB)	2	1290...	10	compute#machineType	16	10240	7680	n1-standard-2-d
0	4 vCPUs, 15 GB RAM	4	1290...	10	compute#machineType	16	10240	15360	n1-standard-4
1	4 vCPUs, 15 GB RAM, 1 scratch disk (1770 GB)	4	1290...	10	compute#machineType	16	10240	15360	n1-standard-4-d
0	8 vCPUs, 30 GB RAM	8	1290...	10	compute#machineType	16	10240	30720	n1-standard-8
1	8 vCPUs, 30 GB RAM, 2 scratch disks (1770 GB, 1770 GB)	8	1290...	10	compute#machineType	16	10240	30720	n1-standard-8-d

Figure 5.8: Manager's metadata

After the task is received by the manager, it will look for the suitable resource available from this metadata, then start a new virtual machine with specific properties on the specific cloud provider, then it will send the task to this particular virtual machine. Another agent of the manager will monitor the execution of these tasks on these resources, and will send periodic notifications to inform when the task or job execution is completed. After finishing the job execution, the manager will make a request to terminate the virtual machine and de-allocate the associated resources. The integration of the GCS API with AMGCS agents is illustrated in figure 5.9.

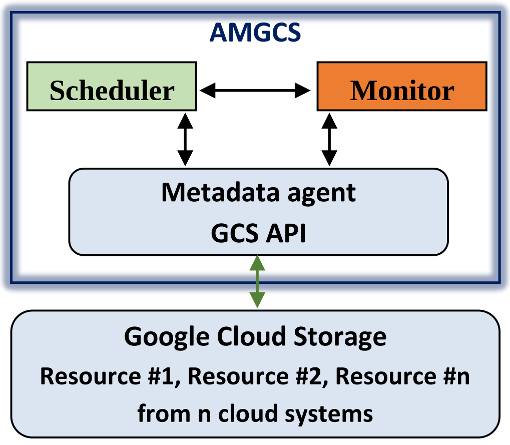


Figure 5.9: Google Cloud Storage integration with AMGCS

Chapter 6

Evaluation and Comparison Study

This chapter discusses the experiments using the AMGCS System to measure its performance and evaluations. It also explains the results in tables and graphs.

6.1 Introduction

To evaluate AMGCS system, we made various experiments to measure its efficiency. We tested the execution of complex jobs that need high compute to be executed. If a compute-intensive job has been requested to be executed with specified desires selected by the user like performance, cost or reliability, the AMGCS manager selects the proper resource from the metadata and sends the job to that resource in order to execute it. This process requires having full information about the resources available in the grid clouds, which is done through our manager by calling the API functions associated with each cloud in the grid cloud. There is an updated list of this information in the manager's metadata.

6.2 Test cases and results

With a centralized manager, the AMGCS has been tested on a compute-intensive job, which is a big matrices multiplication job. These matrices are huge and need long time to be manipulated. The size of the first matrix is [4096][2048] and the second matrix size is [2048][2048], this calculation would take long time to be done, depending on the type of the machine that executes the job, memory, location, server type and so on. The first test case is a single job executed on single cloud system, with the required user desires: *Low cost* and *minimum execution time*. If a regular user wants to perform this job on a cloud system, he will just select any cloud with any properties, as he is looking for a low cost, he might manually select the lowest-cost resource to execute this job. In contrast, the AMGCS manager will select the most proper resource that suits the user desires, and achieves this job efficiently. If we consider that the user has sent the job arbitrary to a lowest-cost resource, which is a virtual machine with a shared core, the job of multiplying these huge matrices took about 43.8 minutes to be calculated. AMGCS manager, however, submitted this job to a more proper VM from the list of resources information available in the manager's metadata, also a shared core but with capabilities that make this VM the best option to select from the available resources in the grid cloud, "*Memory* and *ServerImage*". Figure 6.1 shows the result of this test case.

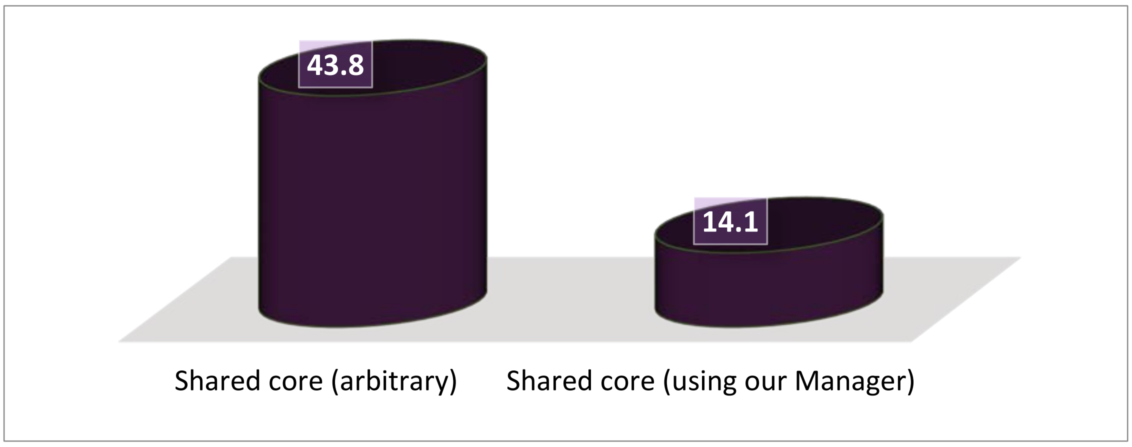


Figure 6.1: Comparison of execution time (minutes), shared core

The optimized execution on the right of figure 6.1 has been achieved by executing the job on a VM with proper properties (capabilities), because the manager knows the full details about all resources from the metadata, it chose one cloud of the grid clouds and create a proper virtual machine to execute this job. The configuration of this particular virtual machine customized the memory, and the server image (*Debian 7 Wheezy)*.

Obviously, here one server is better than the other and hence the significant difference in execution time between them. The server type of the left one is a Windows server, and the type of the optimized one on the right is a Linux server. To make sure that the enhancement here is achieved by the manager's selection strategy and not by the type of the server (Linux or Windows), we did a second test to arbitrary execute the same job on a Linux server also with a shared core, but without using our manager. The results proved that the AMGCS resource scheduling is the reason for that significant enhancement. This is because of the many options that can be customized to a particular virtual machine and hence make this virtual machine the best proper option to execute the required tasks, unlike the regular user's selection that may ignore any consideration to the capabilities or properties of the server's virtual machine. Figure 6.2 shows the execution time on the other server type (Linux) without using AMGCS, it was almost near the time taken on the Windows server in figure 6.1.

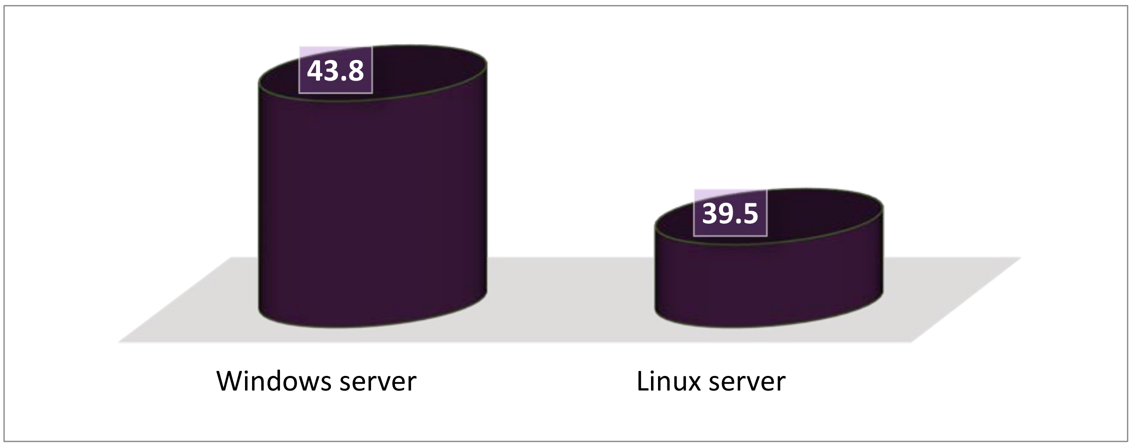


Figure 6.2: Execution time (minutes) without AMGCS, shared core

A third test case, the same job but different user desire, which is *minimum execution time.* Here the user does not care about the cost and the most care about the execution time. It also tested by submitting this job arbitrary to any cloud with any properties, and compare this with the selection of our manager which depends on knowledge of the user desires, job structure and nature, resources available and the recommended resources for complex jobs. As we need the minimum execution time, we definitely need a high compute power to solve this problem as fast as possible. Hence, a virtual machine with eight cores is the proper one. Yet, even with eight cores, we can optimize the performance further by taking into account other factors that might degrade the performance and efficiency of the execution, like the memory and server type. So here the job has been executed on multiple VMs with the same cores number, eight cores.

Figure 6.3 shows the difference in execution time; the one on the right is much less in execution time compared to the arbitrary selected virtual machine on the left. This is because the manager has submitted the job to a more suited cloud with better virtual machine capabilities (*serverType* and *memory* space).

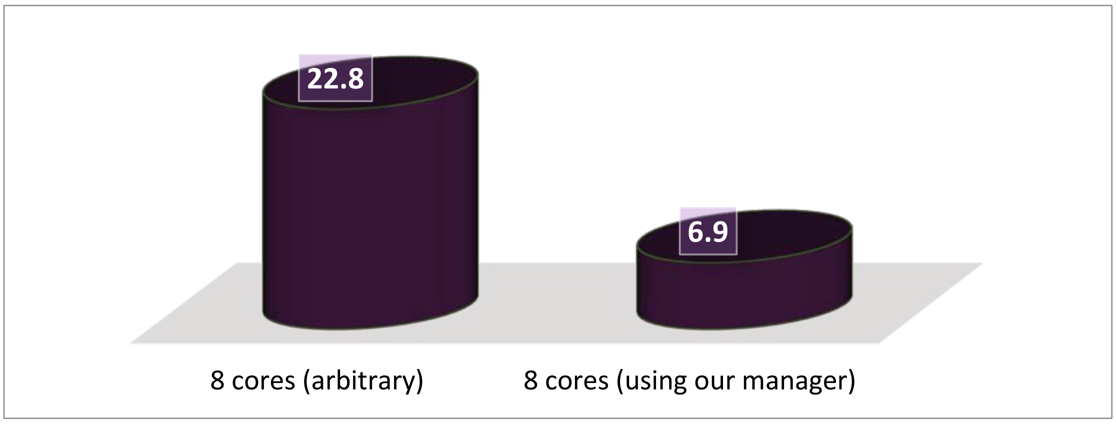


Figure 6.3: Comparison of execution time (minutes), 8 cores

All these test cases are submitting the job to the clouds' virtual machines without decomposing it into tasks. Now if we tried to decompose this big job into tasks, and then execute these tasks individually on multiple clouds. This definitely will increase the performance and the reliability of executing this job.

The fourth test case is to measure the improvement of enhancement after decomposing a job, the same job has been divided into two tasks (parts) to be executed on the grid cloud system, with one user desire which is *minimum execution time*. Decomposition here is programmed in the code just to test the prototypal manager, by dividing the first matrix by half and keep the second as it is, to maintain the matrix multiplication rules. Now the manager has many options to execute these tasks; one of these options is to send each of these tasks to a different virtual machine in order to be executed separately and then combine the results together. This is the case here, where the two tasks of the multiplication job are processed on multiple clouds; hence the time decreased by half.

Figure 6.4 below shows the significant difference in time compared to the execution on single cloud system.

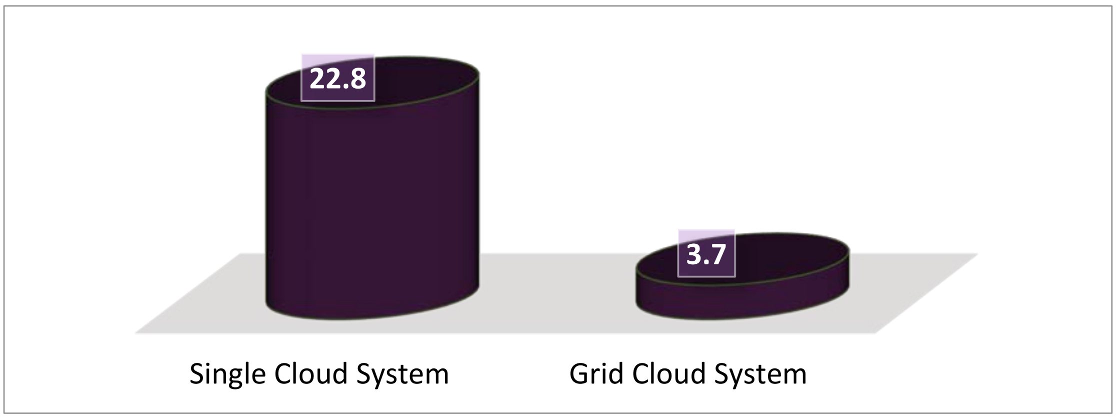


Figure 6.4: Comparison of execution time (minutes), Single Cloud vs. Grid Cloud

The fifth test case was conducted to evaluate the improvement of using grid clouds in executing tasks, by sending replications of these tasks (parts of the job) to multiple clouds. Each task has been replicated and processed two times on multiple virtual machines (other than previously used VMs) so if any failure occurs in any VM we have another copy on another VM. Hence, we guarantee the *reliability* in execution of these tasks, despite the cost that might be high, because here *reliability* is the user desire and reliability is always costly. Figure 6.5 shows the results of this experiment.

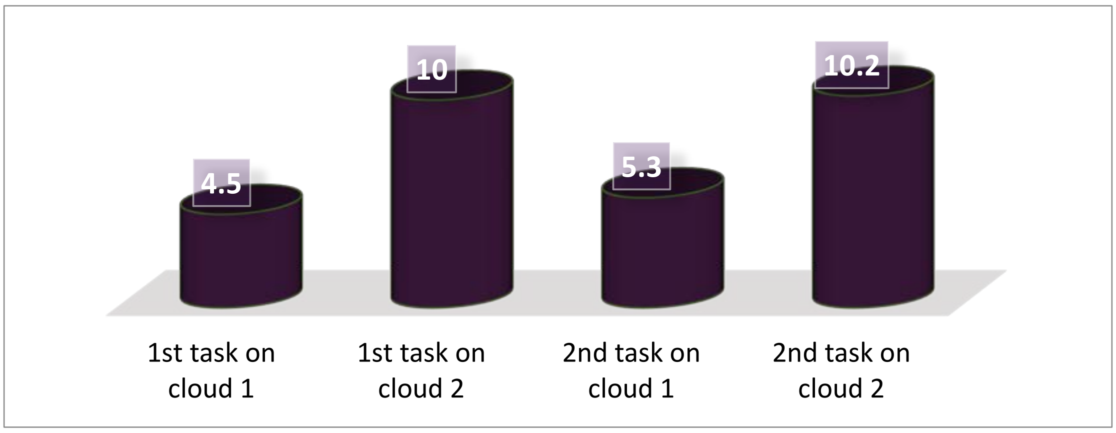


Figure 6.5: Execution time (minutes) for tasks of the job, on Grid Cloud

We end up with an enhancement in executing complex-tasks on grid cloud resources in an efficiently-managed way, combining the comparisons above proves that there is an improvement by 16% - 30% between single and grid cloud, illustrated in figure 6.6.

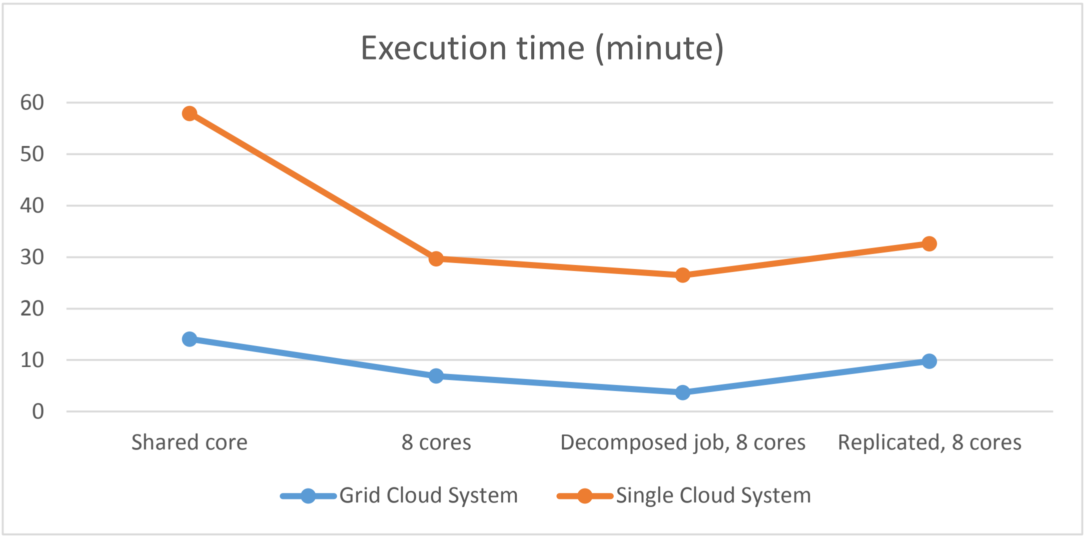


Figure 6.6: Overall enhancement, AMGCS vs. Single Cloud

6.3 Discussion of Experiments Results

Depending on the results that we got from the performed tests and experiments, we can summarize our finding as follows:

- The Manager solves current challenges of executing tasks on the cloud, utilizes grid clouds' resources, solves complex and compute-intensive tasks, and tasks that require high reliability and high performance.
- AMGCS manages jobs and resources and gives good performance in terms of execution time, resource utilization and system throughput compared with using a single cloud system.
- Increasing the number of the grid clouds in the system gives more optimize options and high performance compared to using a small number of grid clouds.
- The overall enhancement of Grid Cloud System is 16% - 32% compared to a single Cloud System.
- The manager does not require any provider-side agreement, only configuring the libraries of the grid clouds.
- Fault tolerance is guaranteed by replication, and increased performance through scaling resources to accommodate user's needs, more or less.
- There is a trade-off between high reliability and cost, our Manager may replicate tasks on multiple clouds and hence more cost.

.

Chapter 7

Conclusion and Future Work

This chapter presents the conclusions and gives the directions for future work. The first section reviews the obtained research results and highlights the main contributions. The second section points out few future research directions.

7.1 Conclusion

In this thesis, we introduced an agent-based manager for grid cloud system (AMGCS) that has been designed based on software agent to ensure platform independency, heterogeneity handling and high flexibility of managing grid clouds. AMGCS has been designed, implemented and successfully tested on real clouds. The benefits of AMGCS are shown in increasing and optimizing the available computing power, managing jobs and resources, and utilizing the grid cloud IaaS resources using the integration between system’s modules and clouds’ APIs.

The research of this thesis can be beneficial to research centers to solve real-world complex problems that need high computing capabilities, such as Bioinformatics applications, engineering simulations and mathematical analysis.

AMGCS manages jobs and resources with good performance by selecting the proper resource for the job, and utilizes available executer nodes in an efficient way.

7.2 Future Work

Future work includes developing a system to run a job on multiple executer nodes at the same time by decomposing the job into a set of tasks, where each task runs on a different machine or uses different resources. In addition, we want to design a tool that is used to read source code of a job and divides it to multiple sub-jobs depending on job nature.

We would also like to support and execute parallel applications on the system. We want to add more services to increase security in the system such as applying security principles in sending and executing job files.

We want to investigate other strategies to distribute framework components and balance load over available resources considering factors such as locality of resources and runtime metrics.

LIST OF REFERENCES

[1] Vladislav Falfushinsky, Olena Skarlat, Vadim Tulchinsky, "Cloud computing platform within Grid Infrastructure", Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2013 IEEE 7th International Conference on (Volume:02), Sept. 2013.

[2] Eugen Feller, Louis Rilling, Christine Morin, "Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds", Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, May 2012.

[3] Guan Le, Ke Xu and Junde Song, "Dynamic Resource Provisioning and Scheduling with Deadline Constraint in Elastic Cloud", Service Sciences (ICSS), 2013 International Conference on, April 2013.

[4] Java Agent DEvelopment Framework. [online] Available at: http://jade.tilab.com/ [Accessed: 1 May 2014].

[5] Baker, M. Buyya, R. and Laforenza, D. (2002), "Grid and grid technologies for wide-area distributed computing," *Software: Practice and Experience,* vol. 32, no. 15, pp. 1437 – 1466.

[6] Foster, L. and Kesselman, C. (2003), "*The grid: blueprint for a new computing infrastructure*", chapter 2, pages 15–52. Series in Computer Architecture and Design. Morgan Kaufmann, 2nd edition, December 2003.

[7] Buyya, R. Chapin, S. and DiNucci, D. (2000), "Architectural Models for Resource Management in the Grid", Grid 2000, Bangalore, India.

[8] ML, B. YA, D. and Gómez, E., "Grid characteristics and uses: a grid definition", First European across grids conference, Santiago de Compostela, Spain, February 2004.

[9] Foster, Ian, and Carl Kesselman. "Computational grids." Cern European Organization for Nuclear Research-Reports-Cern (1998): pp. 15–52.

[10] Chervenak, A. et al. (2001), "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", *Journal of Network and Computer Applications,* Vol. 23, 2001, pp. 187-200.

[11] Mowbray and Miranda, “How web community organization can help grid computing”, International Journal of Web Based Communities, 2007, v3, PP. 44-54.

[12] Berman F, Hey T (2004) The scientific imperative, Chapter 2. In: Foster I, Kesselman C (eds) “The Grid: Blueprint for a New Computing Infrastructure”, 2nd ed. Morgan Kaufman, San Francisco, CA.

[13] Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, Stoica I, Zaharia M, “Above the Clouds – A Berkeley View of Cloud”, Technical report UCB/EECS-2009-28, EECS Department, University of Berkeley, California, 10 February 2009.

[14] Foster I, Zhao Y, Raicu I, Lu S (2008) Cloud Computing and Grid Computing 360-Degree Compared. In: Grid Computing Environments Workshop (GCE’08). doi:10.1109/GCE.2008.4738445.

[15] Miller M (2008) Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online. Que Publishing, Indianapolis.

[16] Menken I (2008) Cloud Computing – The Complete Cornerstone Guide to Cloud Computing Best Practices, Emereo.

[17] Reese G (2009) Cloud Application Architectures. O’Reilly Media, Sebastopol, CA.

[18] Zhang, Miranda and others, (2013) "Investigating Techniques for Automating the Selection of Cloud Infrastructure Services." INTERNATIONAL JOURNAL OF NEXT-GENERATION COMPUTING 4.3.

[19] Stanoevska-Slabeva, Katarina; Wozniak, Thomas; Ristol, Santi, "Grid and cloud computing: a business perspective on technology and applications", Springer, 2010.

[20] Beaty, Donald, "Cloud computing 101", ASHRAE Journal, ISSN 0001-2491, Oct. 2013, Volume 55, Issue 10, p. 88.

[21] Jha S, Merzky A, Fox G, “Clouds Provide Grids with Higher-Levels of Abstraction and Explicit Support for Usage Modes”. Presentation for Open Grid Forum (OFG) 2008. http://www.ogf.org/OGF23/materials/1272/grids_hla_cloud.pdf. Accessed 1 May 2014.

[22] José C. Cunha and Omer F. Rana, "Grid Computing: Software Environments and Tools", ISBN: 978-1-84628-339-0, Springer 2006.

[23] Gopal S. and Sachin U., “A Survey of Software Agent and Ontology”, International Journal of Computer Applications, ISSN 0975-8887, 02/01/2010, Volume 1, Issue 7, pp. 18 – 23.

[24] M. Wooldridge, “An Introduction to Multiagent Systems”, second ed. John Wiley & Sons, 2009.

[25] Gopal S. and N.M. Shelke “A New classification Scheme for Autonomous Software Agent”, IAMA’09, 2009, IEEE Int. Conf., India.

[26] K. M. Sim, "Agent-Based Cloud Computing", IEEE Transactions On Services Computing, VOL. 5, NO. 4, December 2012.

[27] A. Lonea, D. Popescu, and O. Prostean, “A survey of management interfaces for eucalyptus cloud,” in Applied Computational Intelligence and Informatics (SACI), 2012 7th IEEE International Symposium on, May 2012, pp. 261 – 266.

[28] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, “Comparison of open-source cloud management platforms: Openstack and opennebula,” in Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, May 2012, pp. 2457 –2461.

[29] L. Xu and J. Yang, “A management platform for eucalyptusbased iaas,” in Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, Sept. 2011, pp. 193 –197.

[30] C. Baun, M. Kunze, and V. Mauch, “The koala cloud manager: Cloud service management the easy way,” in Cloud Computing (CLOUD), 2011 IEEE International Conference on, July 2011, pp. 744 –745.

[31] C. Baun and M. Kunze, “The KOALA cloud management service: a modern approach for cloud infrastructure management,” in Proceedings of the First International Workshop on Cloud Computing Platforms, ser. CloudCP ’11. New York, NY, USA: ACM, 2011, p. 1:1–1:6.

[32] “Scalr,” [online] Available at: http://github.com/Scalr/. [Accessed: 1 May 2014].

[33] “Apache libcloud,” [online] Available at: http://libcloud.apache.org/. [Accessed: 1 May 2014].

[34] “jcloud,” [online] Available at: http://www.jclouds.org/. [Accessed: 1 May 2014].

[35] “Apache deltacloud,” [online] Available at: http://deltacloud.apache.org/. [Accessed: 1 May 2014].

[36] Yi Wei and M. Brian Blake, "Adaptive Service Workflow Configuration and Agent-based Virtual Resource Management in the Cloud", Cloud Engineering (IC2E), 2013 IEEE International Conference on, March 2013.

[37] Jung G and Sim K. M., “Agent-based Adaptive Resource Allocation on the Cloud Computing Environment”, 2011 International Conference on Parallel Processing Workshops.

[38] Metsch T., Edmonds. A., et al. Open Cloud Computing Interface Core and Models, Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN) 2011.

[39] Venticinque S., Tasquier L., Di Martino B., “Agents based Cloud Computing Interface for Resource Provisioning and Management”, 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems.

[40] Kang J. and Sim K. M., “Towards Agents and Ontology for Cloud Service Discovery”, International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2011.

[41] Domenico Talia, “Cloud Computing and Software Agents: Towards Cloud Intelligent Services”, WOA, volume 741 of CEUR Workshop Proceedings, page 2-6. CEUR-WS.org, (2011).

[42] ZJ Li, Chen C. and Wang K., “Cloud Computing for Agent-Based Urban Transportation Systems”, IEEE Computer Society, 2011.

[43] Haresh M V, Saidalavi Kalady and Govindan V K, “Agent Based Dynamic Resource Allocation on Federated Clouds”, in Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE.

[44] Del Castillo, Lorenzo and others, 2013. "OpenStack Federation in Experimentation Multi-cloud Testbeds." HP Laboratories.

[45] Kurze, Tobias, et al. "Cloud federation." CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization. 2011.

[46] Rawat, S. and Rajamani, L. (2009), "Experiments with CPU Scheduling Algorithm on a Computational Grid ", IEEE International Advance Computing Conference (IACC 2009), PP. 71-75.

[47] Chunlin, L. Xiu, Z . and Layuan, L. (2009), "Resource scheduling with conflicting objectives in grid environments : Model and evaluation " , Journal of Network and Computer Applications 32 , PP. 760 – 769.

[48] Schopf, Jennifer M. “Ten actions when grid scheduling” Grid resource management. Springer US, 2004. 15-23.

[49] Sumathi, G. and Gopalan, N. (2006), “Status Monitoring System for Heterogeneous Grid Environments”, Proc. of 14th IEEE Int. Conf. on Advanced Computing and Communication (ADCOM 2006), 2006.

[50] Huang , K. et al (2009), " Adaptive Processor Allocation with Estimated Job Execution Time in Heterogeneous Computing Grid " , 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications , PP.664-665.

[51] R. Buyya et al., “Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility,” Future Generation Computer Systems, vol. 25, no. 6, pp. 599- 616, June 2009.

[52] K.M. Sim, “Towards Complex Negotiation for Cloud Economy,” Proc. Int’l Conf. Advances in Grid and Pervasive Computing (GPC ’10), R.S. Chang et al., eds., pp. 395-406, 2010.

[53] K.M. Sim, “Towards Agent-Based Cloud Markets (Position Paper),” Proc. Int’l Conf. E-CASE, and E-Technology, pp. 2571-2573, Jan. 2010.