# CHAPTER 1 INTRODUCTION

Over the past several years, there has been a tremendous growth in the area of resource sharing in the computational world. Before the invention of the Internet, end users can only perform computation activities on a stand-alone computer. With the advancement in the microprocessor chip technologies, more and more computational intensive activities are been performed on powerful workstations. Subsequent to the introduction of the Internet, the computational activities are performed using remote servers. A number of users can logon to a server simultaneously to conduct their studies using the client-server architecture. Such demands led to the introduction of the distributed system. A distributed system is defined as hardware or software components that are located at networked computers with communication and coordination done through message passing (Coulouris et al., 2005).

While the demands for greater computation activities led to the introduction of distributed systems, the growth in distributed system further drives more complex computational activities. The Message Passing Interface (MPI) is introduced to allow the parallel execution of computation activities. MPI allows parallel execution in computer cluster, Shared-memory Multi Processor (SMP) as well as supercomputer. MPI also reduces the time required to complete the computation activities.

Buyya (1999) defines a cluster as a type of parallel and distributed system, consisting of a collection of inter-connected stand-alone computers which forms a single integrated computing resource. However, most of the computational activities can only be conducted

in the Local Area Network (LAN); there is lack of technology and network bandwidth to support those activities in different regions or across the Wide Area Network (WAN).

The emergence of high speed networks and the increasing number of computing resources has revolutionized the ability to use distributed resources located worldwide. Grid Computing has been introduced as the solution to coordinate the sharing of resources across different regions. Grid Computing makes computing and services ubiquitous by enabling the aggregation of distributed resources running on multiple platforms in different networks. In addition, Grid Computing provides a transparent platform to execute compute intensive as well as data intensive applications across different regions. The introduction of Grid Computing has changed the way people deal with information.

In line with the popularity of Grid Computing, Web Services have become another promising technology for distributed computing due to the popularity of the Internet. Web Services use open standards and protocols and provides the solutions for different software applications running on different platforms and frameworks to interoperate. In addition, Web Services allow the reuse of services and components within an infrastructure.

Due to above characteristics, Service Oriented Architecture (SOA) is introduced in parallel with Grid Computing to allow the integration of Web Services and Grid (Foster and Kesselman, 2004). SOA is a new model in the distributed system that is extensible and compatible with legacy systems. SOA is a set of principles that defines an architecture that is loosely coupled and comprised of service providers and service consumers that interact according to a negotiated contract or interface (Mansukhani, 2005). SOA provides the

standard that guarantees the interoperability among the services in the Grid environment and simplifies the integration of heterogeneous systems.

The application of the service oriented computing paradigm on Grid Computing has transformed most of the Grid functionalities into services. Furthermore, the service-oriented Grid paradigm offers the potential to provide a fine-grained virtualization of the available resources which will significantly increase the versatility of the Grid (Smith et al., 2004). Another important area that is gaining momentum, along with Grid Computing, is Software as a Service (SaaS). SaaS allows software providers to offer software as services, simplify installations and provide a centralized management. At the same time, SaaS enables end users to access services anytime and anywhere. This leads to Utility Computing where services are being sold to the end users.

The convergence of Grid Computing, SaaS and Utility Computing has led to a new technology, known as Cloud Computing. Cloud Computing is a new paradigm that is able to provide large amount of computing capacity with increase of scalability, availability and fault tolerance. Cloud Computing represents the merging of technologies, software developments in the Internet (Web Services), resources and applications. This emerging technology is able to streamline the on-demand provisioning of software, hardware and data as a service. In summary, the adoption of Cloud Computing as a new technology will benefit both the service providers as well as the end users.

## 1.1 Motivation

Although Cloud Computing provides many new opportunities to the computational world, this technology is still immature. There are still various issues and challenges that have to be addressed especially in the area of job scheduling (Buyya et al., 2002), (Doulamis et al., 2007), (Lee and Zomaya, 2007), (Dong and Akl, 2009), (Fatos et al., 2009), (Ferretti et al., 2010) etc. Job scheduling is a core area of research in Grid and Cloud Computing. Since both the Grid and Cloud Computing need to be able to manage large group of resources and Cloud Computing evolved from Grid, job scheduling play a similar role in both technologies.

Cloud Computing has attracted the attention of the researchers from different fields (Vouk, 2008), (Xu et al., 2009), (Fito et al., 2010), (Reig et al., 2010), (Takabi et al., 2010) etc. Due to the increasing number of virtual distributed resources, designing an effective scheduling algorithm in Cloud Computing is getting more challenging. Moreover, the problems with job scheduling in Cloud Computing are dynamic due to the heterogeneous nature of Grid and Cloud in terms of applications and resources.

Although the job scheduling problem is NP-complete (Grama et al., 2003), there are still many methods to optimize the overall performance. The accuracy of the job length as well as the resource information will determine the efficiency of the scheduling algorithm. Besides, the ability to predict resource and service availability will be the key factors that influence the overall performances.

Grid and Cloud Computing must have the capability to provide services to many users simultaneously and meet different users QoS requirements. The QoS requirements such as availability, scalability, deadline, cost and security are decisive for service selection and resource mapping. In order to address the above issue, further studies and analyses are carried out in the areas of QoS scheduling in Grid and Cloud Computing.

## 1.2 Objectives

The objectives of this thesis are as follows:

- To propose job scheduling algorithm that maximizes resources utilization and minimizes the makespan in Grid and Cloud environment.
- To enhance job scheduling algorithms to maximize reliability and profit while guaranteeing the users QoS requirements.
- To optimize the performance of the proposed job scheduling algorithms through testbed testing.

## 1.3 Scope

The work scopes for this thesis include:

- The studies on the technologies used for Grid and Cloud Computing such as Web Services, SOA, Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), Universal Description, Discovery and Integration (UDDI) and REpresentational State Transfer (REST).
- The understanding of the requirements needed for job scheduling process as well as QoS in Grid and Cloud Computing.

- Defining and developing the testbed environment. The testbed environment includes cluster servers, application servers, virtual servers and desktop computers.
- Selecting and specifying the jobs used for verifying the testbed. The jobs chosen are compute intensive jobs and each job consists of multiple independent tasks. The tasks used are classified as parametric tasks. A parametric task uses the parameter values to determine the task length and output.
- Conducting test on the testbed testing using different job length and job arrival distributions.
- Analyzing the testbed and algorithm performance based on the selected metrics which include makespan, reliability and cost.

### 1.4 Research Methodology

This thesis is conducted using the empirical research method. The research begins with a study on the fundamental concepts of Grid and Cloud Computing which includes the architecture, characteristics, and problems. Subsequently, the relevant technologies such as Web Services and SOA are explored. Based on the study, job scheduling and QoS support have been identified as the two major problem areas. Specifically, there are 2 fundamental issues which are: 1) How to provide an efficient job scheduling decision? 2) How to guarantee the users QoS requirements in Grid and Cloud environment?

Once the major problems are identified, the problems are described using mathematical modeling. Subsequently, various algorithms and mechanisms are proposed and applied to the mathematical models to determine the effectiveness in resolving the problems and achieving the objectives set out.

A Grid and Cloud testbed is designed and developed to conduct the testing of the proposed algorithms using various test cases. The proposed algorithms are then evaluated using empirical testing through the testbed. Also, performance analysis is conducted between the proposed and existing algorithms and enhancements are made to optimize the performance of the proposed algorithms.

### 1.5 Thesis Organization

This thesis is organized into six chapters as follows.

Chapter 1 gives an introduction and overview of the goals of the thesis.

In Chapter 2, a general background of Grid and Cloud Computing is reviewed. This includes the discussion of the underlying architectures, characteristics and problems of the Grid and Cloud Computing. Then, the Web Services and SOA are discussed. Finally, the research opportunities in Grid and Cloud Computing are presented.

Chapter 3 presents the solution that used to address the job scheduling problem in Grid and Cloud environment. The first section describes the scheduling process in Grid and Cloud Computing. Following that, the strengths and weaknesses of the existing job scheduling algorithms are identified. A HSA is proposed and is followed by the discussion of the automatic deployment mechanism. The final section describes the experiment setup and evaluates the performance of the proposed algorithm.

Chapter 4 presents the enhancement of the HSA using benchmarking and adaptive mechanism. The chapter begins with a discussion on the job length estimation using application and resource benchmarking. Then, the ASA is proposed to enhance the HSA. Finally, the experiment setup is presented and the performance of ASA is evaluated.

Chapter 5 presents the solution that guarantee end users QoS requirement and cost management in Grid and Cloud environment. Firstly, the QoS issues in Grid and Cloud Computing are investigated. This includes the discussion on the QoS metrics and SLA. Then, an AQoSSA and the rescheduling mechanism are presented. Subsequently, the performance of AQoSSA is evaluated using the experimental setup running on Grid and Cloud testbed.

Chapter 6 summarizes the efforts and contributions made. This chapter also provides suggestions for future research.

# CHAPTER 2 PRELIMINARY

This chapter reviews the relevant background information on Grid and Cloud Computing. The first section describes the underlying concepts, characteristics and problems in Grid Computing. The second section discusses the system architectures, characteristics and issues in Cloud Computing. A discussion on Web Services and SOA is included as well. The final section presents the research opportunities in Cloud Computing.

## 2.1 Grid Computing

The term Grid Computing originated in the early 1990s as a metaphor for making computer power as easy to access as an electric power grid. Built on top of the Internet and the World Wide Web (WWW), Grid Computing is an infrastructure that enables resource sharing and collaborations within the scientific communities. The origins of these ideas were brought together by Ian Foster, Carl Kesselman, and Steve Tuecke (Foster and Kesselman, 1999).

Over the years, there are many definitions for Grid Computing. Foster and Kesselman (1999) first defined Grid Computing as a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. This definition is redefined to encompass the resources sharing and problems solving in a dynamic, multi-institutional virtual organizations (Foster et al., 2001). Baker, Buyya and Laforenza (2002) defined Grid Computing as a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at

runtime depending on the availability, capability, performance, cost, and users' quality-of-service requirements.

Foster and Kesselman (2004) have suggested that Grid Computing is summarized as:

- Grid coordinates resources that are not subjected to centralized control. The resources are distributed in different administrative domains.
- Grid uses standard, open, general-purpose protocols and interfaces.
- Grid allows its resources to be used to deliver various QoS to meet different user demands.

Grid Computing has emerged as an active platform for large scale scientific and engineering applications. Grid Computing allows a number of collaborative organizations and academics to share documents, software, scientific data, journals as well as hardware and controller. Generally, Grid Computing is broadly classified as Computational Grid and Data Grid.

### 2.1.1 Computational Grid

Computational Grid is a set of hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities (Foster and Kesselman, 1999). This infrastructure provides users with easy, inexpensive and pervasive access to resources.

### 2.1.2 Data Grid

Over the last decade, disk storage capacity is increasing at a faster rate than Moore's Law of processing power (Abbas, 2003). Storage accesses and retrievals have become a great concern due to the huge amount of datasets being generated. Furthermore, scientific analytical activities require access to Terabytes of data that is distributed across different locations. As such, Data Grid infrastructure is created to provide easy data sharing and optimal data transfer performance for data-intensive applications.

Chervenak et al. (2000) defined Data Grid as a technology that provides infrastructure for accessing, transferring, and managing large datasets stored in distributed repositories. Data Grid provides secure access to remote data resources such as flat file, relational and streaming data. Besides, Data Grid also provides an infrastructure in which shared data, storage, networking, and compute resources is delivered to data analysis activities in an integrated, flexible manner (Foster and Kesselman, 2004). In order to access this huge amount of data, various data handling mechanisms are needed. These mechanisms include data discovery, data integration, data mining and data delivery.

### 2.1.3 Grid Computing Architecture

Grid Computing has become the solution for the sharing of distributed computing power and data. Figure 2.1 depicts the layered Grid Computing architecture.

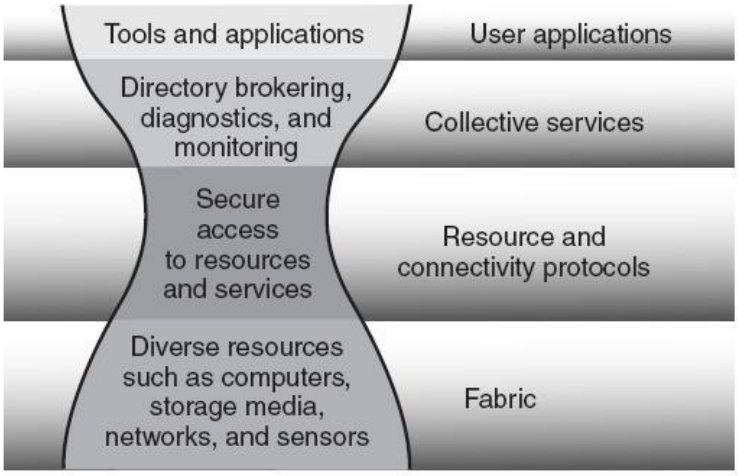


**Figure 2.1: Layered Grid Computing architecture (adapted from Foster and Kesselman, 2004)**

The first layer in Grid Computing architecture is the fabric layer which consists of the physical resources that are shared within the Grid infrastructure. These include computational, storage and network resources. The second layer is the connectivity and resource layer. The connectivity layer defines core communication and authentication protocols that are required for Grid-specific network transactions. The resource layer uses the connectivity layer protocols to control secure negotiations, monitoring, accounting, and payment for the function sharing of individual resources.

The third layer is the collective layer which is responsible for global resource management and interaction with collections of resources. This layer provides directory services, co-allocation, scheduling and brokering services, monitoring and diagnostics services, and data replication services. The final layer of Grid Computing architecture comprises the user applications that operate within the Grid infrastructure.

The Grid architecture provides the functionalities needed for resource sharing and coordination. All the layers are interrelated and dependent on one another. The design of the Grid architecture has made the Grid unique over other distributed systems.

### 2.1.4 Characteristics of Grid Computing

In Grid Computing, there exist many unique characteristics such as heterogeneity, dynamic resource performance, sharing, scalability, adaptability, transparency and security.

#### *2.1.4.1 Heterogeneity*

Grid Computing enables the creation of Virtual Organization (VO) that interconnects geographically distributed heterogeneous computing systems with a variety of resources. The resources comprise of personal computers, servers, clusters, supercomputers, Shared-memory Multi Processor (SMP), data storages, instruments, software applications, and network devices. Each of these resources has a range of capabilities and performances.

#### *2.1.4.2 Dynamic Resource Performance*

VO can span across few small corporate departments that are in the same physical location to large groups of people from different organizations that are spread across the globe (Abbas, 2003). Due to the above nature, the number of available resources changes dynamically. Resources may join or leave from different VOs and existing resources may become idle or busy. Besides, the utilization of the resources changes randomly as the resources are not dedicated to any one application. Resources are utilized to execute multiple Grid jobs as well as local jobs. In addition, the performance of the Grid jobs are always impacted by the local jobs as the latter are given higher priority.

#### *2.1.4.3 Sharing*

Grid Computing encompasses a large variety of concepts involving the sharing of resources such as data exchange and direct accesses to remote software, sensor, etc. Thus, each resource provider shall define the resource management policies, usage patterns and the accessing right for users to view the shared resources.

#### *2.1.4.4 Scalability*

Scalability is clearly a critical necessity for Grid Computing as Grid Computing has to deal with a large number of resources. Grid Computing should be able to scale both upwards and downwards and able to function according to different system scales. Also, the Grid infrastructure should be able to scale well without performance degradation.

#### *2.1.4.5 Adaptability*

Adaptability refers to the degree of possible projected adjustments in practices, processes, or structures of systems or the actual changes of their environment (Andrzejak et al., 2004). The Grid middleware should be able to adapt to changes in the applications or user needs without human intervention. Besides, this middleware must be able to recover whenever a failure occurs.

#### *2.1.4.6 Transparency*

Grid middleware provides a wide variety of users with a transparent access to various types of resources via a single and easy-to-use interface. Grid middleware should support local and remote transparency with respect to invocation and location. Grid middleware should

provide a mechanism to act on behalf of the user at different sites. It should allow users to discover access and process relevant content anywhere in the Grid environment.

#### *2.1.4.7 Security*

Grid Computing involves all four aspects of security which are confidentiality, integrity, authentication and non-repudiation (Coulouris et al., 2005). The security requirements within the Grid environment should be able to support scalable, dynamic and distributed VOs. According to Foster and Kesselman (2004), the security should support single sign on, delegation support and interoperability with local security rules.

### 2.1.5 Problems of Grid Computing

Many researchers have proposed various mechanisms to address the problems related to Grid Computing. One of the biggest challenges in Grid Computing is to design an efficient job scheduling algorithm to cater for the heterogeneous nature of Grid. The heterogeneity of the Grid is represented by various parameters, including resources, types of applications and optimization objectives (Dong and Selim, 2007). This heterogeneity has resulted in different capabilities for job processing, data access and data transfer. Besides heterogeneity, the availability of computation resources, storage as well as network bandwidth also affects the scheduler's decision making.

In addition, modern scientific experiments usually have extreme performance demands and capacity requirements. Such requirements have caused a tremendous increase in the number of resources needed and further aggravate the potential performance degradation. Besides, the probability of resources failures in Grid Computing is high due to the

heterogeneity nature and dispersion of resources across multiple VOs (Foster and Kesselman, 2004). Furthermore, most of the scientific applications are complex and involved many interacting activities. These applications are prone to errors and failures. Thus, the Grid infrastructure should provide a mechanism to detect faults and recover from faults.

The sharing of resources across different VOs has resulted in a wide variety of problems related to security (M.Nithya and Banu, 2010). Firstly, each resource has its own policy and procedure. The problem becomes more complicated due to different requirements from service providers and end users. A security policy is required to address this problem without impacting the usability of the resources or create security loop holes in the existing systems.

Although Grid Computing enables the sharing of distributed resources to execute various types of application, not all applications are able to utilize this advantage. The existing Grid infrastructure only supports specific scientific applications which are executed in limited number of locations. A technology must be defined in order to allow the applications to run in anywhere and at any time. The next section presents how Grid Computing evolves and an introduction to a new trend of computing.

### 2.1.6 Grid Evolution

Grid Computing has evolved from the earlier developments in small scale administrative domain to a large scale multiple administrative domains connected to one another. When Grid Computing was introduced, the infrastructure was primarily used by the research

communities. As the technology become mature, Grid Computing has provided significant business values. Since the resources are distributed in different VOs, there is a need for a common set of Application Programming Interfaces (APIs), protocols as well as services to support the business needs. One of the popular architecture that provides the standard APIs and protocol is the Open Grid Services Architecture (OGSA).

*2.1.6.1 Open Grid Services Architecture (OGSA)*

OGSA defines a standard architecture for integrating and managing services within VOs. Foster et al. (2002) defined OGSA architecture as the standard mechanism for creating, naming, and discovering transient Grid service instances. OGSA is built based on the concepts and technologies from both the Grid Computing and Web Services communities. OGSA provides location transparency and protocol bindings for service instances and supports integration with underlying native platform facilities. This has allowed businesses to build Grid infrastructure across the enterprise and business partners.

OGSA Grid technologies are based on SOA. Figure 2.2 shows the principal elements of OGSA. The main elements of OGSA are Open Grid Services Infrastructure (OGSI), OGSA services and OGSA schemas.

OGSI defines the mechanisms for creating, managing and exchanging information among the Grid service instances. OGSI provides the specification used by the clients to deal with the instances. OGSA services include core services, data services and computation management services. These services provide the standard features such as service discovery, security, messaging, data accessing, provisioning and resource management and

services deployment. OGSA schemas provide the standard schemas for messaging and communication between different service components (Foster and Kesselman, 2004).

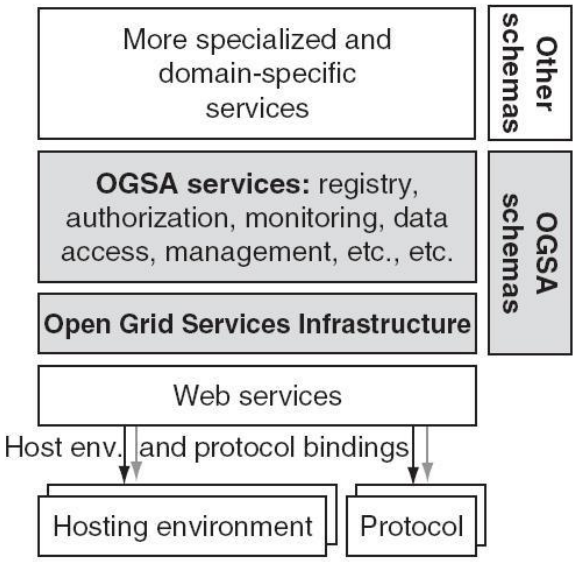


**Figure 2.2: Principal elements of OGSA (adapted from Foster and Kesselman, 2004)**

The use of SOA in OGSA provides a fine-grained virtualization of the available resources. This would significantly increase the versatility of a Grid Computing. In addition, the service-oriented Grid provides a binding element among Grid specific services at the hardware and application services level (Stanoevska-slabeva et al., 2009).

*2.1.6.2 Utility Computing*

The maturity of OGSA has led to the creation of Utility Computing. Multiple services are offered by third party providers and users can use the resources in a pay-per-use manner. Organizations could access the Utility Computing for a specific business function. Any task requirements that exceed the in-house infrastructure capacity could be submitted to the Utility Computing. This saves the organization from making additional infrastructure investments.

*2.1.6.3 Software as Service (SaaS)*

Together with Grid Computing, SaaS popularity is gaining momentum. The idea of sharing software through the Internet is not a new phenomenon. Traditional software is developed to run on the end user computers. With SaaS, users can access the software by using a Web browser, Secure Shell (SSH) or through API call. Users are not required to invest in the infrastructure or purchase a copy of the license to run the software. OGSA has the potential to provide the necessary flexibility and scalability architecture for SaaS.

*2.1.6.4 Virtualization*

In line with the software development, the popularity and adoption of virtualization has increased tremendously. Virtualization was first introduced and popularized by IBM in the 1960s for running multiple software contexts in mainframe systems. Virtualization allows abstraction and isolation of lower level functionalities and underlying hardware (Vouk, 2008).

Multiple Virtual Machines (VMs) can run on a single physical server through hypervisor. Hypervisor performs the abstraction of the hardware for each individual VMs. Besides, hypervisor provides VMs with unified and consistent access to the CPU, memory, and I/O resources on the physical machine. Since multiple VMs are executed on the same physical machine, the resources are utilized more efficiently. This is due to the multiple core processors where the cores are not fully utilized within a single machine environment.

Basically, there are two different types of hypervisor-based virtualizations as shown on Figure 2.3.

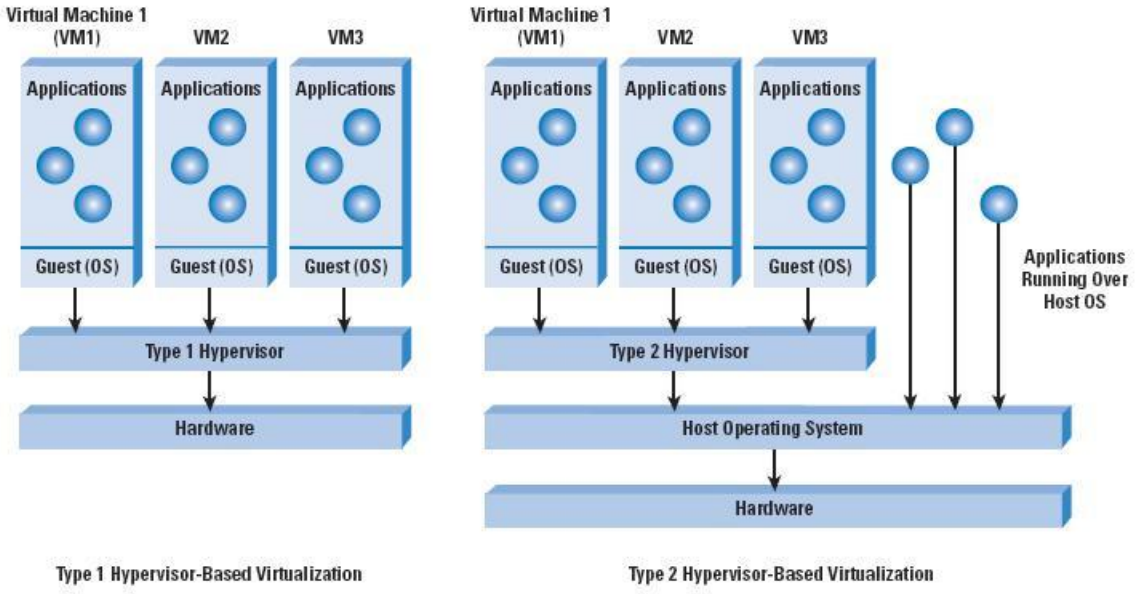


**Figure 2.3: Types of hypervisor-based virtualizations (adapted from Sridhar, 2009)**

The first type of hypervisor based virtualization allows the virtualization to run directly over the hardware. Over the past few years, processor manufacturers such as AMD and Intel, have provided strong hardware support for virtualization. These include the Intel Virtualization Technology (Intel® VT) and AMD-V™ Technology. This has increased the resources performances as compared to running virtualization over an operating system.

The second type of hypervisor based virtualization allows the virtualization to run on top of operating system. However, in such an implementation, the performance is not as good as running the virtualization directly on the hardware. Usually, this type of implementation requires more memory when compared to the first type of implementation.

Virtualization provides the abstractions for the fabric to be unified as a pool of resources and the resource overlays such as application services are built on top of them (Foster et al.,

2008). In addition, virtualization allows the aggregation of the physical resources to improve the resources utilization and flexibility to adapt to changing requirements and workloads. Since virtualization allows run time configurations, the virtualization is adapted and managed more effectively in the distributed and heterogeneous environment.

In addition, virtualization improves service availability by providing strong support during failover. In a virtual environment, the nodes are backup and migrated without service interruption. This directly increases the application service availability. Moreover, virtualization also improves responsiveness as common resources are cached and provided based on demand anytime and anywhere.

The convergence of SOA and Grid Computing has given rise to the availability of many software applications in the form of Grid services. At the same time, Utility Computing and SaaS have gained a lot of momentum recently. Similarly, virtualization technology is starting to gain popularity among organizations. The convergence of above technologies has led to a new computing trend, Cloud Computing.

## 2.2 Cloud Computing

Cloud Computing has been a dominant research topic for the past several years. Cloud Computing is believed to be able to provide reliable services delivered through data centers. Moreover, recent advances in virtualization technologies have led to the popularity of Cloud Computing. Virtualization technology has become more adoptable with the presence of multi-core processors. Also, virtualization can ease the deployment of applications on multiple resources.

There have been many definitions for Cloud Computing. Ian Foster et al. (2008) defined Cloud Computing as a 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. Virtualization of the hardware or software resources allows these resources to be added or withdrawn according to demand. These resources are accessed by the end users through a standard interface.

Buyya et al. (2009) defined Cloud Computing as a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources. The resources are provisioned on demand and have to meet the service-level agreements that are established between the service provider and consumers. These resources will be accessible as a composite service via Web Service technologies.

Armbrust et al. (2009) from Berkeley RAD Lab defined Cloud Computing as both the applications delivered as services over the Internet, and the hardware and system software in the datacentre that provide those services. The services are referred as SaaS and the datacentre hardware and software is referred as Cloud.

A Cloud environment can comprise of a single Cloud or multiple Clouds. A single Cloud environment is classified into public Cloud and private Cloud. A public Cloud provides service to end users via Internet as pay-as-you use manner, much like utilities. Usually, the datacentre hardware and software is operated by third parties such as Google and Amazon.

Public Cloud is usually hosted away from customer premises. A private Cloud provides services to a limited number of people behind a firewall. The datacentre hardware and software is fully owned by a single company who has total control over the applications running on the infrastructure. Private Cloud provides companies a high level of control over the Cloud resources.

In a multiple Clouds environment, hybrid Clouds combine both public and private Clouds and enable the end users to execute applications on an internal as well as external Cloud infrastructure. Hybrid Clouds are normally used to handle workload overloading whereby applications are deployed to the public Cloud whenever the private Cloud is overloaded. However, hybrid Cloud is unsuited for the applications that require large amount of data transfers to the public Cloud but only require a small amount of processing.

In summary, organizations can choose to deploy applications on public, private or hybrid Cloud. These Clouds offer different benefits and organizations can implement the best Cloud that fulfils their needs.

### 2.2.1 System Architecture

Figure 2.4 depicts the system architecture of Cloud Computing. The three layers architecture of Cloud Computing consists of Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Cloud Computing provides services at these three different layers.

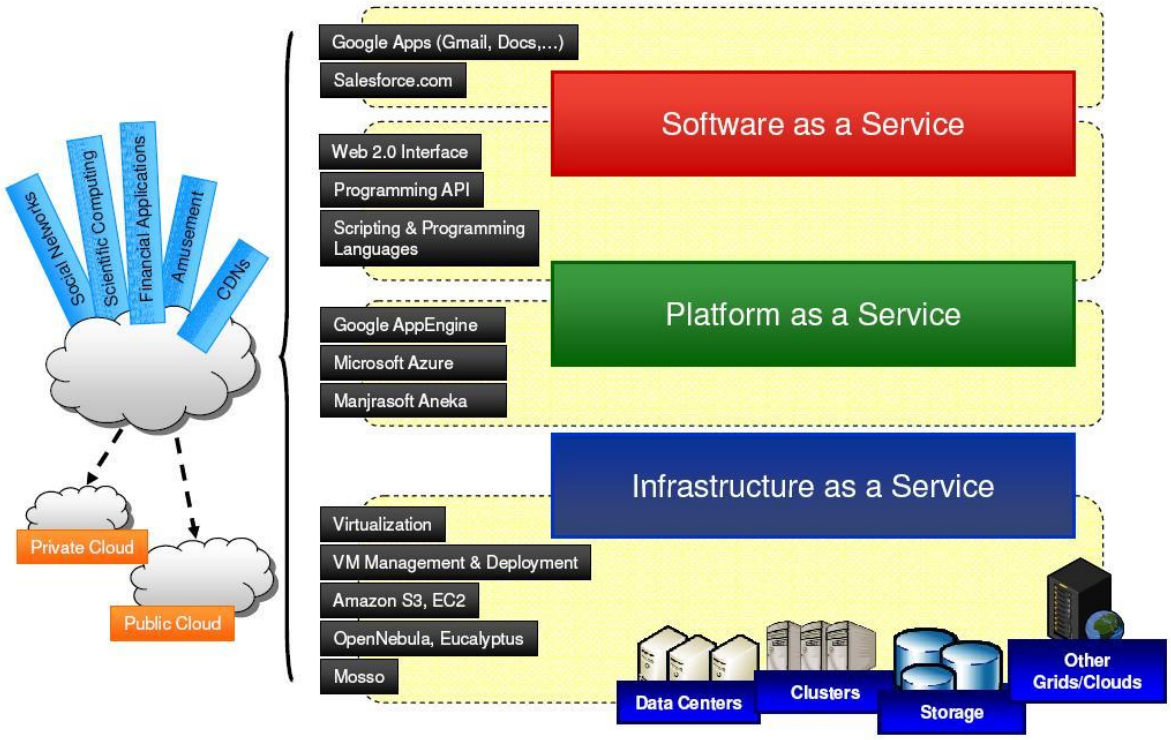


**Figure 2.4: Cloud Computing System Architecture (adapted from Vecchiola et al., 2009)**

*2.2.1.1 Infrastructure as Service (IaaS)*

IaaS is the layer that provides basic storage, computation capabilities and network infrastructure as standardized services. However, this layer is different from the fabric layer of Grid infrastructure since IaaS providers offer the resources as services like utilities. In addition, the resources are provided as a unified resource through virtualization which the users can access through a standard interface (Foster et al., 2008). Figure 2.5 shows the Cloud architecture in relation to Cloud services. Some examples of the IaaS are Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage Service (S3), OpenNebula and Joyent.

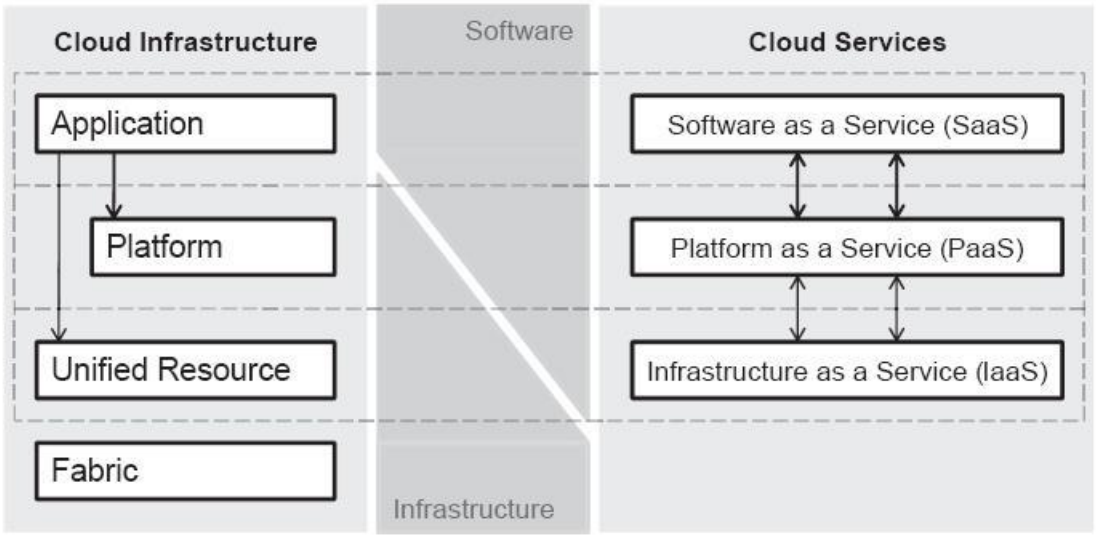


**Figure 2.5: Cloud architecture related to Cloud services (adapted from Foster et al., 2008)**

Amazon EC2 provides resizable compute capacity in the Cloud environment and provides complete control of the computing resources to the end users. End users can rent servers for a certain CPU speed, memory, disk capacity together with the OS and applications. By using virtualization technologies, Amazon EC2 allows end users to reserve resources for a specific time and only charges base on the capacity used (Amazon, 2010a).

Amazon S3 is the storage service for the Internet. Amazon S3 provides a simple Web Services interface that can store and retrieve any amount of data, at any time and from anywhere in the Web. The objects are redundantly stored on multiple devices across multiple facilities in an Amazon S3 region. In addition, Amazon S3 provides checksum verification on all network traffics to detect data packets corruption during data storage retrieval. Amazon S3 stores any amount of data inexpensively and securely, while ensuring that the data will always be available (Amazon, 2010b).

OpenNebula is an open source, virtual infrastructure manager that deploys virtualized services on a local pool of resources and external IaaS Clouds. OpenNebula enables the dynamic deployment and replacement of virtualized services within and across sites. By using the driver based architecture, OpenNebula is integrated with multiple virtual machine managers, transfer managers and external Cloud providers (Sotomayor et al., 2009).

*2.2.1.2 Platform as Service (PaaS)*

PaaS is an abstraction layer between the software applications (SaaS) and the virtualized infrastructure (IaaS). PaaS provides a software platform on which users can build their applications and host them on the PaaS provider's infrastructure (Sridhar, 2009). This layer provides a development framework to build, test, and deploy custom applications according to the specifications of a particular platform. The commercial example of PaaS includes the Google Apps Engine, Microsoft Azure and Manjrasoft Aneka (Stanoevska-slabeva et al., 2009). However, the migration of existing applications to a PaaS environment is difficult. This is because the applications need to follow the APIs and be written in specific language provided by the PaaS infrastructure.

Google App Engine is the PaaS infrastructure that allows developers to build and deploy Web applications running on Google's infrastructure. Google App Engine provides a set of APIs that allow developers to integrate with services provided by Google such as Mail, Datastore and Memcache. In addition, Google App Engine is scalable in terms of network traffic as well as data storage. The applications running on Google App Engine are written in Java or Python programming languages (Google, 2010a).

Windows Azure is the PaaS provided by Microsoft for developing scalable applications for the Cloud. Windows Azure provides developers with on-demand computation resources and storages to build, deploy and manage their Web applications through Microsoft datacentres. The developers can create services that run on .NET framework using the Microsoft Azure Software Development Kit (SDK). In addition, Windows Azure supports popular standards and protocols including SOAP, REST, XML, and PHP (Windows, 2010).

Manjrasoft Aneka is a pure implementation of PaaS that provides platform and framework for developing distributed applications on the Cloud. Aneka is based on the .NET framework and is designed to utilize the computing power of Windows based machines. The core value of Aneka is its service oriented runtime environment that is deployed on both physical and virtual infrastructures and allows the execution of applications developed with different application models (Vecchiola et al., 2009).

*2.2.1.3 Software as Service (SaaS)*

SaaS is the software that is owned, delivered and managed remotely by software providers or service providers. This software is offered as services in a pay-per-use manner (Wohl, 2008). SaaS is provided through the communication networks and made accessible to users via a Web browser. Instead of purchasing the server licenses for the software, end users can obtain the same functions through a hosted service from service providers through network connection. The most widely known examples are Google Apps and salesforce.com.

Google Apps is the SaaS provider that allows end users to access their software through the Internet. The software provided by Google Apps includes Google Mail, Google Calendar,

Google Docs, Google Groups and Google Video (Google, 2010b). Salesforce.com is another SaaS provider for enterprise Cloud Computing providing hosted software services such as Sales $Cloud_2$ and Service $Cloud_2$ (Salesforce.com, 2010).

*2.2.1.4 Benefits*

As summarized, IaaS, PaaS and SaaS offer different types of services to service providers as well as end users. In order to derive the maximum benefits from these services, service providers, developers and end users must understand their requirements and design their applications wisely.

The first advantage of deploying applications in Cloud Computing is to minimize the risk of deploying a physical infrastructure. With Cloud Computing, organizations can deploy their new software products in the Cloud to the level of success without any infrastructure investment. If the applications proof to be successful, the organization can subscribe for more resources. The organization does not need to worry about the scalability issue as this is handled by the service providers. Furthermore, for specific events, organization can rent more resources to handle workload spikes at a lower cost.

The second advantage of Cloud Computing is the reduction of execution and response time. Cloud Computing can shorten the time taken for executing batch jobs since the jobs are distributed across multiple resources. Also, Cloud Computing can help to optimize the response time by minimizing the execution time for CPU intensive tasks.

Service providers can develop many applications as services through the standard interface over a Cloud. Service providers can generate revenues by providing services as well as the physical infrastructure based on the pay-per-use model. Besides, all the installations, maintenance, upgrading and patching are performed at the datacentre. This eases the job of service providers by eliminating the needs to visit client sites for installation, upgrading and troubleshooting.

Cloud Computing allows start-up organizations to begin their business at a low cost by allowing these organizations to rent the resources without paying for the full cost of purchasing the hardware infrastructure and software licenses. Indirectly, the lower entry cost increases the pace of innovation since more small capital companies can be involved.

### 2.2.2 Characteristics of Cloud Computing

Cloud Computing has some close connections to other relevant technologies such as Grid Computing, Utility Computing, Cluster Computing and Distributed Systems (Foster et al., 2008). Figure 2.6 shows an overview of the relationship between Cloud Computing and other domains. The figure shows that Cloud Computing lies within Web 2.0 which supports the SOA while Grid Computing overlaps with supercomputers, clusters as well as Clouds and is less scalable when compared to supercomputers and Clouds.

Although Cloud Computing evolved from Grid Computing and relies on Grid Computing as its backbone and infrastructure support, there is a set of characteristics that distinguishes Cloud Computing from Grid Computing. Foster et al. (2008) have identified the differences

between Grid and Cloud Computing in various aspects such as security, programming model, compute model, data model, application and abstraction.

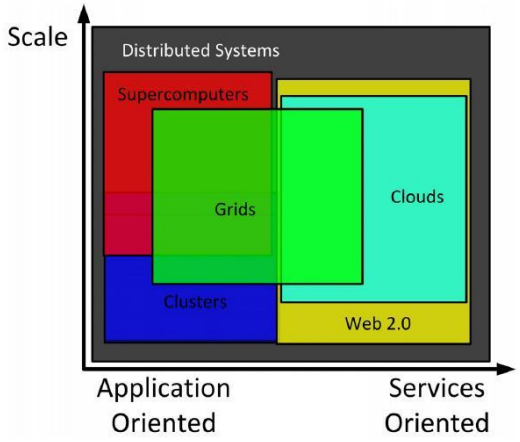


**Figure 2.6: Relationship of Cloud Computing and other domains (adapted from Foster et al., 2008)**

*2.2.2.1 Virtualization*

Cloud Computing provides different services using virtualization and storage technologies. Zhang and Zhou (2009), defined two basic approaches to enable virtualization in Cloud Computing which are, hardware virtualization and software virtualization. The hardware virtualization is used to manage the hardware resources in a plug-and-play mode. Hardware equipment is added or removed without affecting the normal operations of other equipment in the system. Software virtualization allows the use of software image management to enable software sharing. Software images are created from a set of software systems including, operating system, middleware, and applications. Besides, software virtualization also allows dynamic code assembly and execution. Code elements are dynamically retrieved and executed based on the composition of reusable code elements and just-in-time complier technologies.

*2.2.2.2 Scalability*

Cloud Computing is evolved from Grid Computing and Cloud Computing relies on Grid Computing as its backbone and infrastructure support. Thus, Cloud Computing must be scalable across different regions, hardware types as well as software configurations. Cloud Computing can run on physical resources as well as on virtualized new resources. The virtualized resources are adaptable to various jobs' requirements from different users. This scalability and flexibility is driving the emergence of the Cloud Computing.

Mei, Chan and Tse (2008) classified Cloud scalability into horizontal and vertical. Horizontal Cloud scalability is the ability to connect and integrate multiple Clouds to work as one logical Cloud. For instance, a Cloud providing computational services can request services from another Cloud that provides Calculation services. Vertical Cloud scalability is the ability to improve the capacity of a Cloud by providing more resources in the Cloud. The resources are provided dynamically on demand.

*2.2.2.3 Accessibility*

Cloud services are accessed using simple and pervasive methods such as standard Web Services framework or APIs with Internet. The users do not need to remember the complex command as required in Grid Computing. The services can be access as Utility Computing. Through Cloud Computing, users are able to access the services anywhere, anytime, share data and store their data safely. A more equitable pay-per-use scheme may be implemented to charge users based on the combined resources usage.

*2.2.2.4 On-demand Service Provisioning*

Cloud computing requires a dynamic computing infrastructure. A dynamic computing infrastructure is critical to effectively support the elastic nature of service provisioning and de-provisioning as requested by users while maintaining high levels of reliability and security. Thus, the resources must be able to plug into a Cloud environment dynamically. Basically, these resources are not permanent parts of the IT infrastructure. Besides, users can customize their resources requirements and make resources reservations. Different resources will be provided based on demand and users need.

*2.2.2.5 QoS Guaranteed*

Cloud Computing provides services to multiple users in different regions and having different requirements. Thus, Cloud Computing must provide QoS guarantees for different users' requirements that are documented in the SLA. In general, SLA would include non-functional requirements such as hardware availability, service availability, storage, network, performance, costing etc. Furthermore, penalties are imposed in the event of any violation of the SLA. A good Cloud Computing system will be able to meet multiple QoS requirements.

In summary, Cloud Computing and Grid Computing share some common characteristic in the area of architecture, objectives as well as technologies. However, there are also some differences in various aspects such as virtualization, business model, programming model, applications and abstractions. The next section describes the technologies used by Cloud Computing, namely Web Services and Service Oriented Architecture (SOA).

### 2.2.3 Web Services

Web Services are emerging technology in the development of distributed applications in Grid and Cloud Computing. Web Services are software components stored on a single computer but is accessed via method calls by an application on another computer over a network (Deitel and Deitel, 2009). Nowadays, there are many Web Services available across the world that can be used in various fields such as science, engineering, business, education, government etc. The popularity of Web Services is due to the growth of Internet which has enabled users throughout the world to access the Web Services wherever they are located.

Web Services provide service interfaces that enable clients to communicate with servers without human supervision. Web Services have promoted software portability and reusability for applications. Web Services allow complex application to be developed by integrating with other Web Services. In addition, Web Services are independent of any particular programming model which allows the communication between client and server in a platform-independent manner. Hence, Web Services have become the solution for building distributed applications.

Basically, each Web Services have a Uniform Resource Identifier (URI) which is a general resource identifier whose value is either a Uniform Resource Locator (URL) or a Uniform Resource Name (URN) (Deitel and Deitel, 2009). The most common value of URI is URL. URL is a persistent reference containing the domain name of the server and the service to which it refers.

Web Services communicate using XML and Hypertext Transfer Protocol (HTTP). XML is used for data representations and message exchanges between clients and servers. XML is adopted for its readability and ease of debugging. SOAP is used to encapsulate the messages which are then transmitted over HTTP. In addition, each Web Services contain service descriptions which are defined by the Web Services Description Language (WSDL). The description describes the interface as well as others related information. WSDL defines the XML schema such as element names, definition, types, message, interface, bindings and others (Deitel and Deitel, 2009). The WSDL documents are accessed either directly or indirectly using the URIs via a directory service such as UDDI.

Besides SOAP, another alternative approach is to use REpresentational State Transfer (REST). REST is a constrained style of operation, in which client use URLs and HTTP operation to manipulate resources that are represented in XML (Fielding, 2000). REST is suitable for describing the architecture of distributed resource access.

*2.2.3.1 XML*

XML has emerged rapidly as a new approach to deliver structured data over the Web. XML is a widely supported open technology and has become the standard format for describing the data that is exchanged between applications over the Internet.

An XML document contains texts that represent contents and elements that specify the document's structure. The advantage of XML is because XML describes the data in a standardized and structured way, making it both humans and machines readable. It is simple enough for the machine to read and write and yet easily understandable by

developers to interpret and debug the codes. Furthermore, XML is simple to use and there and many XML parsers available.

However, the more important factors are having XML adhering to a standard that allows programs written in different languages on different platforms to communicate with each other and a standardized method to invoke remote resources on the Web for exchanging complex data.

*2.2.3.2 SOAP*

SOAP is the communication protocol used by Web Services to transmit requests and replies between clients and servers (Deitel and Deitel, 2009). SOAP defines a scheme for using XML to represent the contents of request and reply messages. SOAP also defines the communication of documents and the protocol used. The current version of SOAP supports HTTP and SMTP communications.

SOAP specifies the rules of using XML to package messages. SOAP message is packaged into a container known as an envelope which consists of an optional header with a required body. The header is used to establish the context for a service and audit operation and is altered by intermediary devices for routing and security purposes. The body is used to store the document for a particular Web Services. The body element contains the name of the procedure as well as the URI for the relevant service description. The body element is either request message or reply message.

SOAP supports synchronous and asynchronous communication. In synchronous communication, a client invokes a request for a service and then waits for a response. A single port is open to send and receive data. The synchronous communication is suitable for the applications that require an immediate response. Basically, for synchronous communications, the services requests are often process within short time duration. However, synchronous communication is not suitable for services that require longer processing time because synchronous communication will block the client until the output of the Web Services is returned. In asynchronous communication, the client invokes the service but does not wait for the response. Hence, asynchronous communication is suitable for the services that require a significant amount of time to process the request. After making an asynchronous request, the client can carry on with other threads of execution without blocking.

*2.2.3.3 WSDL*

WSDL is an XML-based language for describing Web services. WSDL describes the point of contact for a service provider and provides a simple way for service providers to describe the basic format of the requests. WSDL defines XML schema to represent the components of a service description which include definitions, types, message, interface, binding and services (Deitel and Deitel, 2009).

WSDL service description is separated into two categories; namely, abstract description and concrete description. An abstract description establishes the interface characteristics of Web Services. WSDL includes a set of definitions for the types used by the service and a description of the set of messages exchanged. Also, WSDL groups together the collection

of operations that belong to same Web Services. Each operation represents a specific action performed by the service and the operation consists of a set of input and output messages (Erl, 2005). A concrete description is used for communication. The description comprises the binding and the service. The binding element is used to describe the requirements for a service to establish physical connections. SOAP is the most common form of binding. The service element specifies the name of the service and one or more endpoints where an instance of the service may be contacted.

*2.2.3.4 UDDI*

Universal Description, Discovery and Integration (UDDI) is a standard for registering and searching Web Services within directories. The main function of UDDI is to provide discovery of services. The UDDI is a central directory service where a service provider can publish, register, and search for Web Services and offers the service user a location to look for appropriate service (Bellwood, 2002).

The data stored in the UDDI directory is in XML format and is divided into three main categories; namely, white pages, yellow pages, and green pages. White pages contain general information like name, description, address, etc. about the service providers. Yellow pages contain general classification data on industrial categories based on standard taxonomies. Green pages contain detailed technical information about the Web Services (Overhage and Thomas, 2003).

Figure 2.7 depicts the communication between the Web Services and UDDI. Firstly, service provider will publish their services using WSDL and the definitions will be stored

at UDDI. When the service users look up for a service, queries are sent to the UDDI to locate the service and the query responses are returned to service users. Using the received response, the service users will send request to the service provider in WSDL. Finally, the service providers will response based on the service request.

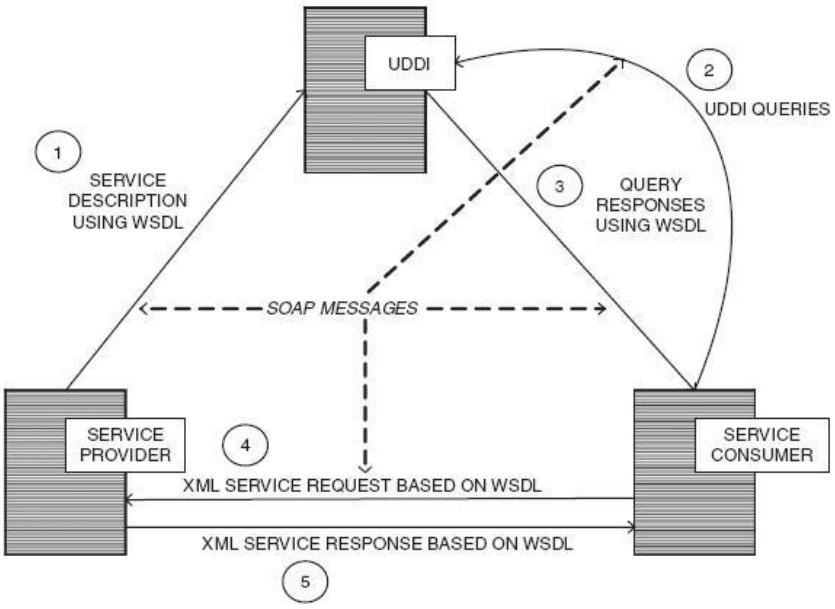


**Figure 2.7: Communication between Web Services and UDDI (adapted from Diamadopoulou et al., 2008)**

*2.2.3.5 REST*

Although most of the Web Services use SOAP as the communication protocol, there is another alternative; namely, REST. Feng, Shen and Fan (2009) defined REST as a coordinated set of architectural constraints that restricts the roles or features of architectural elements and the relationships among those elements within any architecture that conforms to that style. REST is implemented on HTTP protocol and the services using HTTP should conform to the semantics of the Web (Fielding, 2000). REST is not an API but contains a set of guidelines for designing applications to run over HTTP.

With REST, the emphasis is on the manipulation of data resources rather than on interfaces. As long as the interfaces remain unchanged, the client and the server can interact normally. In REST, the resource can be a physical object or an abstract concept while a representation is any useful information about the state of a resource. Thus, REST is a new mode of service abstraction and REST helps to understand the original look of HTTP and fully utilize current Web features. In comparison to SOAP, REST offers a simple style of accessing resources that works with the Web.

In summary, Web Services provide an infrastructure for maintaining a structured form of interoperability between clients and servers. Besides, Web Services also scale well across the Internet. Since HTTP is used to transport SOAP messages, HTTP allows the services to be accessed by different organizations and administrative domains as most of the policies will allow the HTTP traffic to pass through. Due to these natures, Web Services have proven to be the solution for implementing distributed applications across heterogeneous environments.

### 2.2.4 Service Oriented Architecture

The widespread use of the Internet and the introduction of XML as a structured format have triggered the introduction of Service-Oriented Architecture (SOA) (Mansukhani, 2005). Nowadays, SOA is widely used as the architecture to integrate various independent applications into services. SOA enables the provision of a large number of Web Services. Many legacy applications are converted to services based on SOA to adapt to the requirements of modern societies.

SOA is best described as a set of architectural concepts and principles that includes systems development methods, techniques and related technologies that enable the implementation of service-oriented enterprise applications (Feuerlicht, 2010). SOA transforms different functions of a software process in a standardized way into services to support multiple projects implementation. The difference between SOA and traditional application architecture is that SOA emphasizes interface, protocol, communication, coordination, working process, search, cooperation and publication (Wang and Liao, 2009).

#### *2.2.4.1 Characteristics*

SOA is an emerging model of system architecture. Some of the key characteristics of SOA include loosely-coupled model, open standard and service composition (Erl, 2005). SOA supports loosely-coupled interface which minimizes dependencies between services. This allows the services to evolve independently. By implementing standardized service abstraction layers, a loosely coupled relationship is achieved between different applications domain.

Besides, SOA is based on open standards. SOA uses SOAP, WSDL and XML for message representation and communication. The use of the open and standardized messaging model eliminates the needs for underlying service logic to share systems type (Erl, 2005). In addition, SOA composites different services and supports the construction of collaboration service. This allows the composition of business processes into a SOA model and can lead to highly optimized automation environments.

*2.2.4.2 Benefits*

Due to above characteristics, SOA has become an important technology in today's software development. Since SOA is based on open standard, SOA permits the sharing and reuse of services across multiple projects. This has directly reduced the cost as well as time needed to implement a new project. Moreover, SOA addresses the system integration problems by introducing the loosely-coupled model. SOA provides the capability to reconfigure different processes rapidly by selecting the available set of services making it highly adaptable to changes.

The popularity of Web Services has enabled many legacy applications to participate in SOA. With SOA, the isolated applications are interoperated without requiring the development of expensive middleware. This saves cost and effort of integration. Furthermore, SOA supports the composition of services. Developers can develop any solution required by the organization by compositing the services available on the Internet.

Cloud Computing is becoming the emerging technology for business, academy as well as end users. Since SOA allows services to be discovered, composited and executed, SOA play a major role in supporting Cloud Computing. With SOA, all the hardware, software as well as data resources are wrapped as services in Cloud Computing.

## 2.3 Research Opportunities

After the literature studies on the system architectures, characteristics and issues in Grid and Cloud Computing, several important areas of concerns that need to be addressed are

identified. These problems include job scheduling, Quality of Service (QoS) and cost management. These three issues are summarized as below.

### 2.3.1 Job Scheduling

Job scheduling is the one of the main research area in Grid and Cloud Computing (Buyya et al., 2002), (Doulamis et al., 2007), (Lee and Zomaya, 2007), (Dong and Akl, 2009), (Fatos et al., 2009), (Ferretti et al., 2010) etc. Since there is a common need in both Grid and Cloud Computing to be able to manage large group of resources, job scheduling plays a major role in both technologies. An efficient job scheduling technique is required to maximize the resources utilization and minimize the total execution time.

Due to the increasing number of distributed resources and the introduction of virtualization in Cloud Computing, designing an effective job scheduling algorithm in Cloud Computing is very challenging. Although the job scheduling problem in Grid and Cloud Computing has been shown to be NP-complete (Grama et al., 2003), (Armbrust et al., 2009), (Boloor et al., 2010), (Reig et al., 2010), there are still various heuristics methods that are used in this heterogeneous environment to optimize the performance.

### 2.3.2 Quality of Service

As Grid and Cloud computing have emerged as the technologies that provide services to end users at distributed location, the ability to guarantee QoS is becoming critical. As the service providers need to provide services to end users with different QoS requirements, the guaranteeing of QoS in Grid and Cloud environment is becoming extremely challenging (Diamadopoulou et al., 2008), (Li et al., 2009a), (Xu et al., 2009), (Ferretti et al., 2010) etc.

An efficient QoS aware resource allocation mechanism is needed to meet end users QoS requirements.

### 2.3.3 Cost Management

Cloud Computing is a recent technology trend that delivers on demand IT resources on pay-per-use basis. This pay-per-use mechanism is applied to computation, storage as well as network bandwidth. There are various pricing schemes that can be used by the service providers to estimate and determine the cost (Buyya et al., 2009), (Truong and Dustdar, 2010), (Yeo et al., 2010), (Chaisiri et al., 2011). Cost management serves as a basis for managing the supply and demand of the resources. An efficient pricing mechanism will ensure that the service providers achieve higher revenues.

In summary, the next few chapters will provide the solution that are used to address the above issues. A job scheduling algorithm that maximizes resources utilization and minimizes the makespan in Grid and Cloud environment is proposed. Furthermore, this algorithm will be enhanced to maximize reliability and profit while guaranteeing the users QoS requirements.

## 2.4 Chapter Summary

This chapter presented the background information of Grid Computing, Cloud Computing, Web Services and Service Oriented Architecture (SOA). As summarized, Grid Computing is a mature technology that used to coordinate resource sharing and problem solving in dynamic, multi-institutional virtual organizations. Grid Computing has provided an

infrastructure that allows end users to perform their computation activities in multiple heterogeneous resources.

Together with the growth of Web Services and SOA, Grid Computing is offered in the form of Grid services that is accessed anywhere and anytime. This has led to the introduction of Utility Computing where computer resources are accessed on a pay-per-use basis. In line with this, there is an evolution in SaaS due to the increasing interests in network-based subscription software model. The convergence of Grid Computing, Utility Computing as well as SaaS has formed a new technology referred to Cloud Computing. Cloud Computing can provide a flexible, scalable and accessible infrastructure that allow the execution of SaaS. Cloud Computing delivers IT services as computing utilities and these services are dynamically provisioned through virtualization.

Several problems such as job scheduling, Quality of Service (QoS) and cost management have been identified during the studies. This thesis will provide the solution to address the above issues. Chapter 3 will present the solution that addresses the job scheduling problem in Grid and Cloud environment.

# CHAPTER 3 HYBRID SCHEDULING ALGORITHM (HSA) AND AUTOMATIC DEPLOYMENT

This chapter presents the solution that used to address the job scheduling problem in Grid and Cloud environment. The first section details the scheduling process in Grid and Cloud Computing and the second section presents the existing scheduling algorithms. In the third section, the Hybrid Scheduling Algorithm (HSA) and the automatic deployment mechanism are proposed. The final section describes the experiment setup and the evaluations performed on the HSA and the automatic deployment mechanism.

## 3.1 Grid and Cloud Scheduling Process

In terms of characteristics, architecture and technologies, there are many commonalities between Grid and Cloud Computing. One of their common needs is to be able to manage large group of resources effectively. As such, job scheduling is one of the main research areas in Grid and Cloud Computing. In order to coordinate the resources, an effective and efficient job scheduling algorithm is essential. Since Cloud Computing is evolved from Grid Computing and relies on Grid Computing as its backbone and infrastructure support, a deep understanding of job scheduling process in Grid Computing is essential.

Job scheduling in Grid Computing is defined as the process of making scheduling decisions involving resources over multiple administrative domains (Schopf, 2004). Job scheduling is a process that maps and manages the execution of different jobs on distributed resources. Job scheduling involves searching multiple administrative domains to use a single resource or scheduling a single job to use multiple resources at a single site or multiple sites. In

recent years, the number of submitted jobs as well as distributed resources has increased tremendously. In response to this, an efficient scheduling algorithm is needed to coordinate multiple jobs onto multiple resources. A proper scheduling algorithm will have a significant impact on the performance of the overall system.

Figure 3.1 depicts a logical Grid scheduling architecture proposed by (Zhu, 2003). When users submit jobs to the Grid environment, the Grid scheduler processes the jobs and selects the best resource to execute the jobs according to the jobs' requirements as well as the resources information collected from the Grid Information Service (GIS). GIS provides the aggregate information about the resources in each domain as well as a list of supported jobs. This information is very important for Grid scheduler to make a proper scheduling decision.

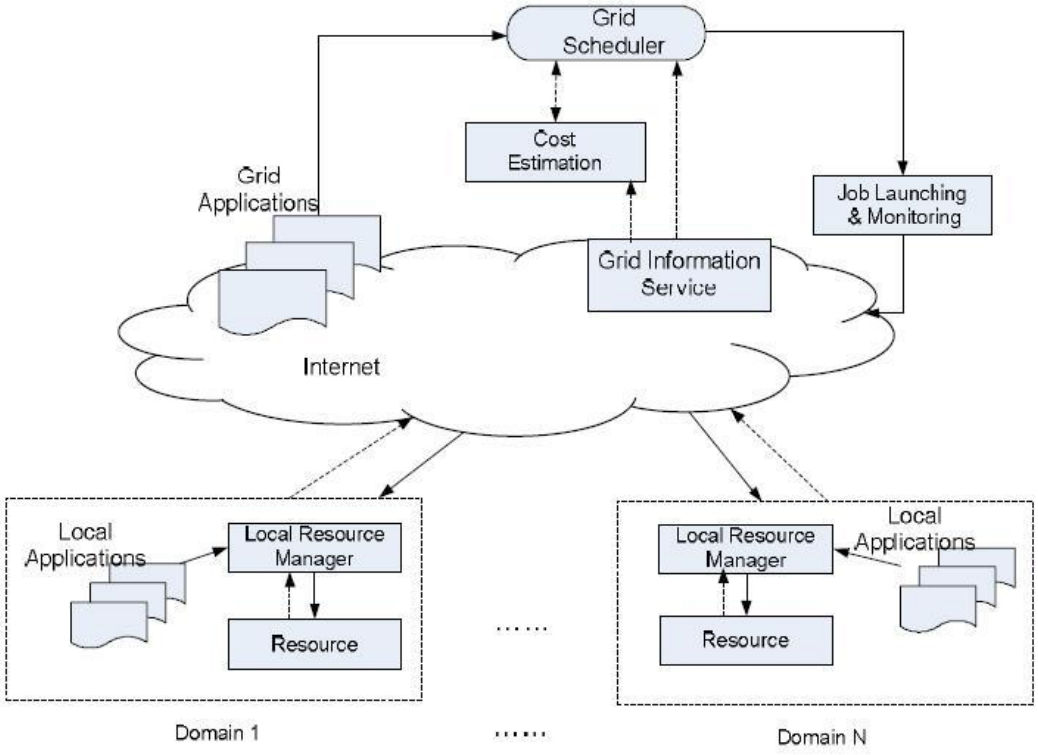


**Figure 3.1: A logical Grid scheduling architecture (adapted from Zhu, 2003)**

Once the Grid scheduler has enough information about the jobs and resources, the Grid scheduler can generate a job-to-resource mapping list. Using this list, the deployer will dispatch the jobs to the resources accordingly. Although the scheduling process looks simple, it is quite challenging in the Grid environment, mainly due to the heterogeneity of the Grid. Since the resources are distributed in multiple administrative domains, Grid scheduler often makes mapping decisions where the scheduler has no control over the local resources. Moreover, the resources information collected from each domain is aggregated information and is often out dated. The resources availability is unpredictable and makes the Grid an opportunistic environment.

In general, Grid scheduling process is divided into three phases; namely, resource discovery, resource selection and job execution. In Grid environment, resource discovery phase discovers the availability and the status of resources that are distributed in multiple locations efficiently. This process generates a list of potential resources. On completion of discovery phase, the resource selection process is used to choose the best resources for each job. Once the job-to-resource mapping list is generated, the jobs are dispatched to the resources for execution. Figure 3.2 depicts the Grid scheduling process.

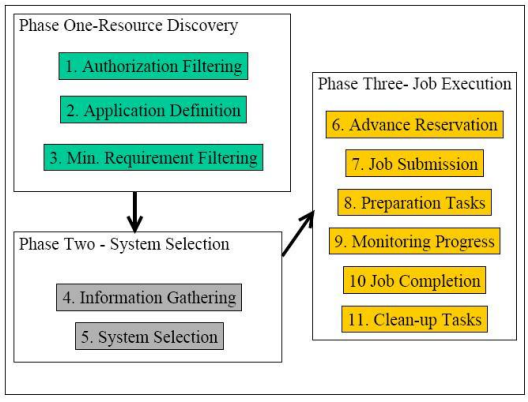


**Figure 3.2: Grid scheduling process (adapted from Schopf, 2004)**

### 3.1.1 Resource Discovery

An efficient resource discovery mechanism is one of the fundamental requirements for Grid scheduling. In this phase, a potential set of resources is determined based on the jobs submitted. The first step of resource discovery is to determine the set of resources that the user submitting the job has access to. After which, the set of potential resources is further filtered based on the job requirements such as processor capabilities, memory requirement, network bandwidth, software availabilities as well as temporary storage required. The resources information is collected from GIS. The role of GIS is to collect and predict the current resource information. GISs are usually organized in a centralized and hierarchical structure. The final step of resource discovery is to filter out the resources that do not meet the minimal job requirements and generate a set of potential resources.

There are various studies on resource discovery; Ranjan, Harwood and Buyya (2008) presented the current state of the art in Grid resource discovery and investigated on the various decentralized resource discovery techniques using Peer-to-Peer (P2P) network model. Figure 3.3 depicts the P2P network model.

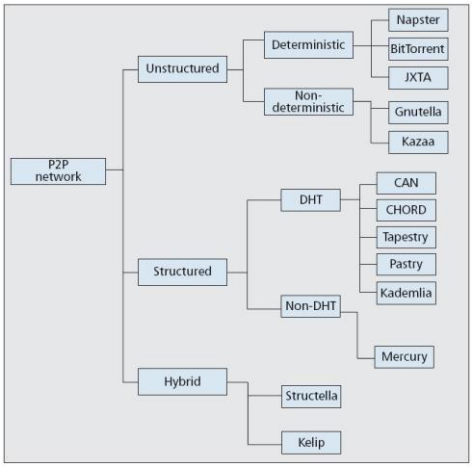


**Figure 3.3: P2P network model (adapted from Ranjan et al., 2008)**

Forestiero and Mastroianni (2009) proposed Antares, a bio-inspired algorithm that is used to construct a decentralized and self-organized P2P information system in computational Grid. The algorithms make use of agents that traverse the Grid through P2P interconnections to replicate and sort similar indexes into neighbouring Grid hosts. This improves the rapidity and effectiveness of discovery operations.

Due to the popularity of Cloud Computing, resource discovery based on Web Services has become very popular. Since Grid services can easily be mapped onto the Web Services, the Grid resource discovery is treated as discovery of the Web Services. Kaur and Sengupta (2007) presented the Web Services based resource discovery mechanism for Grid. The UDDI database is used as the source of resource information to process the resource query. The system uses UDDI rich query model to discover Grid services. In addition, the system consists of an extended version of WSDL to describe the Grid Services. However, the drawback of the system is that the system uses central servers and databases which are prone to bottlenecks and single point of failures.

Molt'o, Hern'andez and Alonso (2008) proposed a SOA based meta-scheduler Grid service that is accessed through the network. The system is developed on top of Globus Toolkit which uses Meta-computing Directory Service (MDS) for the resource discovery. However the system cannot scale well. Li et al. (2008) presented ROSSE, which is a Rough sets-based search engine for Grid service discovery that deal with the uncertain properties in service matching. The research includes a QoS model to filter functionally matched services with their QoS-related non-functional performance.

### 3.1.2 Resource Selection

After a potential set of resources is generated, the best resource must be selected for the scheduled job. In order to generate the most suitable job-to-resource mapping, the detailed information about the resource is needed. This information is used to compute the ranking according to the objective function of the scheduler. The resource with highest ranking is selected as the candidate for the job. The process is repeated until all the jobs are matched. Finally, a job-to-resource mapping list is generated. The objective function of the scheduler is discussed in section 3.2.4.

### 3.1.3 Job Execution

This final phase of Grid scheduling process is job execution. Once the job-to-resource mapping is generated, the job is dispatched and submitted to the selected resources for execution. The job submission may include transferring data and software configuration prior to job execution. With the advancements in virtualization technologies, a customized runtime environment is created and the computational resources are automatically deployed. There are various studies on service deployments using virtualization technology. These include (Lacour et al., 2005), (Sun et al., 2008), (Kecskemeti et al., 2008), (House et al., 2008) and (Lizhe et al., 2009).

Lacour, Perez and Priol (2005) proposed a generic application description model that translates a specific application description into a generic model that allows the automatic deployment of the application. Currently, the model supports CCM and MPICH-G2 applications. Sun et al. (2008) and Kecskemeti et al. (2008) deployed services with virtual appliances. Virtual appliances provide a simple and unified interface to deploy multiple,

interrelated software components into heterogeneous environments. House et al. (2008) proposed an architecture for hosting services on virtual clusters that span across multiple administrative domains. The virtual hosting infrastructure provides software deployment across multiple service providers.

Wang et al. (2009) developed the Web Services based virtual machine provider for Grid infrastructures called Grid Virtualization Engine (GVE). GVE implements a scalable distributed architecture with a hierarchical flavour. Besides, GVE provides the standard Web Services interface for users to manipulate virtual machine resources and supports automatic service deployment.

Once the runtime environment is configured, the job is executed using a single command, multiple scripts or services. During the job execution period, the statuses of the job as well as the resources are monitored. If the job execution is not making sufficient progress, the job may be rescheduled. There are various studies on rescheduling including (Reed and Mendes, 2005), (Hussain et al., 2008), (Netto and Buyya, 2008), (Diaz, 2009), (Lee et al., 2009) and (Zhang et al., 2009).

Reed and Mendes (2005) monitored the performance data to verify whether the contracted specifications are satisfied. If the contract is not satisfied, the system will reschedule the application on a new set of resources that can satisfy the original contract specifications. Hussain et al. (2008) rescheduled threads at run time when the computer resources are having high idle time. Netto and Buyya (2008) rescheduled co-allocation requests in the environments where runtime estimations are inaccurate. The proposed model uses

processor remapping to overcome the limitations of response time guarantees and the needs for fragmentation reduction.

Lee, Subrata and Zomaya (2009) proposed the Adaptive Dual-objective Scheduling (ADOS) algorithm that performs the scheduling by accounting for the completion time and resource usages. The algorithm incorporates rescheduling to deal with unforeseen performance fluctuations. Zhang, Koelbel and Cooper (2009) focused on the workflow scheduling mechanism. The authors proposed a light-weight hybrid scheduling mechanism and a two-step rescheduling decision approach.

Once the job is completed, the status as well as the job output is saved at the user's specified location. Besides, the system will perform the clean-up tasks to remove any temporary settings or pointer related to the job.

One of the biggest challenges in job execution phase is job failure. A Grid job cannot execute before the deadline and is terminated due to resource failures or the presence of local jobs. Advance reservation is one of the methods that are used to address the problem. In this method, part or all of the resources needed are reserved in advance. The reservations may be based on time and may include special cost. Some of the researches in resource reservations are (Kunrath et al., 2008), (Moaddeli et al., 2008), (Rajah et al., 2009) and (Singh et al., 2009).

Kunrath et al. (2008) proposed an algorithm that reserves a certain percentage of the resource capabilities, rather than making a full resource reservation. Since the scheduler has

to deal with time and resource capabilities, variable slots are used to organize the reservations in the schedule. Moaddeli et al. (2008) explored the trade-offs and bottlenecks of incorporating time constrained flexible advance reservation in backfilling methods.

Rajah, Ranka and Xia (2009) designed a novel admission control and scheduling algorithm for bulk data transfer. The algorithm combines advance reservation, multipath routing, and bandwidth reassignment via periodic re-optimization into a cohesive optimization-based framework. Singh, Kesselman and Deelman (2009) implemented strategies by integrating reservations within the resource management to prove the deterministic QoS while addressing fairness issues with regards to best effort services.

### 3.1.4 Differences

Although Grid and Cloud Computing share several commonalities, there are significant differences in the job scheduling process. Since the services are pre-configured in Grid Computing, the resources in Grid can only serve a limited number of requests. In Cloud Computing, the resources are provided on demand. During the deployment process, the required services are automatically configured using automatic deployment mechanism. As a result, Cloud solves the problem of scalability.

In Grid Computing, the requests are processed immediately using the pre-configure services. However, in Cloud Computing, extra time is required to create the environment for executing the requests. The extra time is needed for image and services deployment. The requests are processed after the environment is created.

In Grid Computing, the resource consists of physical servers that are pre-configured. During the rescheduling process, the scheduler is required to discover the resources within the existing pre-configured services. Due to such constraints, there are difficulties in matching the available resources and increase the failure rate to service the requests. Such problem are overcome using Cloud Computing where resources are provided on demand. During the rescheduling process, the same image files as well as the services are deployed to any available resources.

In summary, there are still many active studies in the area of job scheduling in Grid and Cloud Computing. In the next section, the existing job scheduling algorithms are presented.

## 3.2 Scheduling Algorithms

Jobs scheduling plays an important role in Grid and Cloud Computing. A proper job scheduling algorithm is able to increase resource utilizations as well as improving the overall performance in terms of makespan, load balancing and QoS. There are various types of scheduling algorithms in Grid and Cloud Computing. Each scheduling algorithm is different in terms of the structure of the scheduler, type of job requests, time of the scheduling decision, and the objective functions of the scheduling algorithm.

### 3.2.1 Structure of Scheduler

Scheduling structure is divided into centralized scheduling and decentralized scheduling. In a centralized environment, a central scheduler will handle all the job requests. The central scheduler gathers the resources information from multiple sites and makes scheduling decisions based on the jobs and resources information. The central scheduler is easy to

manage and has all the information of the available resources. The only deficiency is single point of failure due to hardware error or network connection. Figure 3.4 depicts the centralized scheduling.

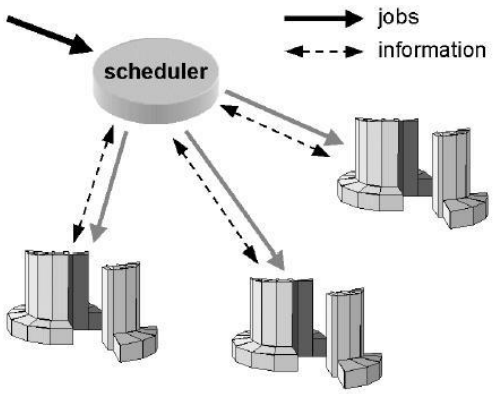


**Figure 3.4: Centralized Scheduling (adapted from Hamscher et al., 2000)**

In a decentralized environment, the job requests are submitted to different distributed schedulers. Each distributed scheduler collects aggregate resources information from their controlled sites and then exchanges the information with the adjacent schedulers. This type of scheduler is more scalable and provides better fault-tolerance and reliability as compared to the central scheduler. However, the distributed schedulers are difficult to manage in terms of job synchronization since the job requests are submitted to different distributed schedulers (Hamscher et al., 2000). Figure 3.5 depicts the decentralized scheduling.

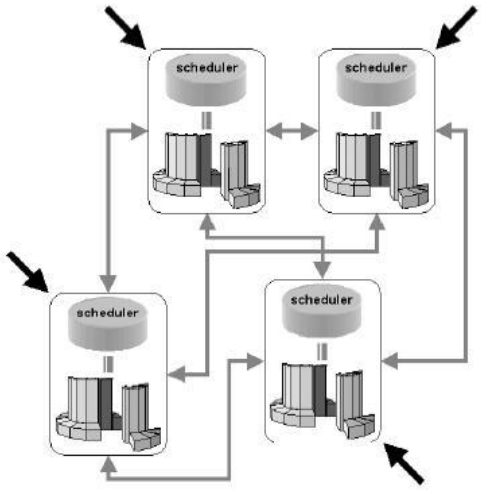


**Figure 3.5: Decentralized Scheduling (adapted from Hamscher et al., 2000)**

Zikos and Karatza (2008) proposed a hierarchical structure that combines the benefits of the centralized and decentralized schedulers. In a hierarchical model, the central scheduler handles all the job requests and collects aggregate resources information from each site at a fixed interval. Then, the central scheduler dispatches the jobs in groups to different sites. The local schedulers at each site will assign the local resources to the jobs.

### 3.2.2 Job

A job is a set of atomic tasks to be executed on a set of resources. Each task may compose of atomic sub tasks (Baker et al., 2002). Before making any scheduling decision, it is essential for the scheduler to take into account the attributes and requirements of the job. The attributes of the job include the number of instructions or job length, communication volume and the subtask relationship. Some of the job requirements are: memory, storage, bandwidth, platform, software and QoS. The job attributes and the requirements are gathered in a process known as application profiling.

#### *3.2.2.1 Job Length*

One of the attributes that is used to estimate the length of a job is to calculate the total number of instructions needed to execute a job. The information is collected either using software or hardware. Zhang et al. (2007) proposed a mechanism that uses the processor hardware counter to provide detail information such as control flow prediction, execution rate and memory access behaviours. This information is useful in profiling the application execution behaviour and the length of a job. The drawback of this approach is the extra processor overhead needed to gather the instructions information. Besides, different mechanisms are needed for the processor hardware counter when running on different computer architecture.

#### *3.2.2.2 Communication Volume*

Communication profiling is very important especially in data intensive distributed applications. Communication profiling provides the statistical information such as communication volume and communication activity of the application. With this information, Grid scheduler can predict the communication time required for different applications.

#### *3.2.2.3 Subtask Relationship*

Parameter sweep job and workflow job are two popular application models in distributed system. Parameter sweep job comprises a set of computational tasks that are mostly independent. These tasks are executed independently and simultaneously. This type of application model exist in many fields of science and engineering, including computational fluid dynamics, bioinformatics, discrete-event simulation, computational biology,

astronomy and computer graphics (Foster and Kesselman, 2004). Due to its nature, parameter sweep job is distributed to multiple computing resources and is suitable for use in a distributed environment.

Workflow job comprises of multiple dependent tasks that are to be executed in a predefined order. The overall task is partitioned into sub tasks that linked to each other in order of precedence. A task cannot begin until its entire parent sub-task is completed. The dependencies are crucial to the design of a job scheduling algorithm.

The workflow job is represented as a Directed Acyclic Graph (DAG) where a node represents a task and a directed edge represents the precedence between its two vertices. Figure 3.6 depicts the DAG. The workflow job may include the transfer of a huge amount of data among each sub-task (Deelman et al., 2009).

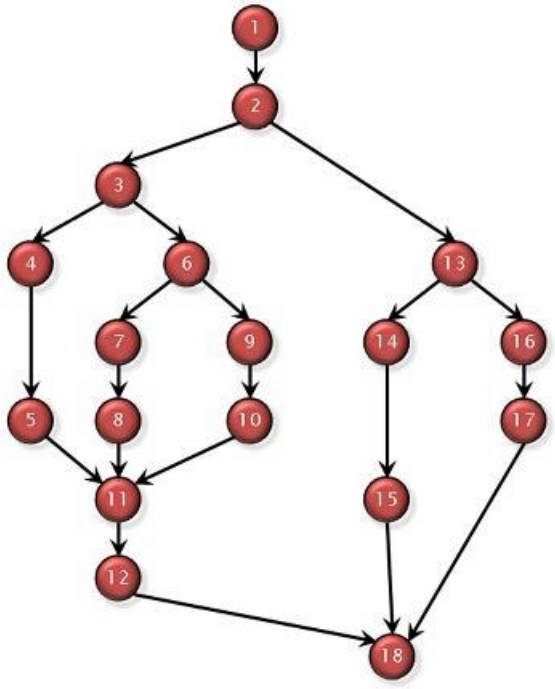


**Figure 3.6: Directed Acyclic Graph (DAG)**

*3.2.2.4 Hardware Requirement*

Different applications require different types of hardware resources such as the number of processors required, the number of cores per processor, the processor speed, the amount of Random-Access Memory (RAM) required, the minimum storage capacity measure in Gigabytes (GB) and the amount of graphical memory. In addition, some applications may require a minimum amount of bandwidth for transferring data.

*3.2.2.5 Software Requirement*

The software requirements include the platform, operating system as well as the applications needed to execute the job. The platform includes Windows, Macintosh, UNIX and others. The operating system includes Windows family, OS X, Solaris, Linux and others. Also, while applications only run on a specified version of the operating systems, others may require the presence of Java runtime environment or .NET framework.

### 3.2.3 Time of Scheduling

When a scheduler receives a job request, there are three different mechanisms to schedule the job request namely, static scheduling, dynamic scheduling and hybrid scheduling.

*3.2.3.1 Static Scheduling*

In static scheduling, the scheduler analyses the resource information, tasks attributes and users' requirements before making a scheduling decision. Once the task has been assigned to the resource, the placement of task is fixed. Static scheduling is easier to program and manage. However, a drawback of static scheduling is that static scheduling is not adaptive. Once the task placement is made, the scheduling decision is fixed regardless of the future

changes to the underlying resources performances that have significant impact on the earlier scheduling decision. The static scheduling algorithms are studied in (Braun et al., 2001), (Baskiyar and Dickinson, 2005), (Vahdat-Nejad and Monsefi, 2008) and (Gallet et al., 2009).

Braun et al. (2001) examined eleven heuristics approaches to optimally map tasks onto the distributed resources in the heterogeneous computing environments. The problem of allocating the tasks to resources in Grid or Cloud Computing is NP-complete. Heuristics approaches are used to improve the scheduling decision as they are more adaptive to the Grid infrastructure where both resources and application are heterogeneous and dynamic. The popular approaches are Min-Min and Max-Min.

In Min-Min, all the tasks in the queue are grouped in a set and then sorted by placing the task in the queue according to the job length with the shortest at the head of the queue. Subsequently, each task is assigned to the resources that can complete the task with the minimum completion time. This process continues until all the tasks in the set are assigned to the resources.

The Max-Min method is very similar to Min-Min except that the tasks are sorted with the longest job length placed first in the queue. Then, the task is assigned to the resources that return the minimum completion time.

Baskiyar and Dickinson (2005) proposed a static scheduling with insertion-based scheduling and multiple task duplication to schedule directed a-cyclic weighted task graphs

on a set of heterogeneous processors. Vahdat-Nejad and Monsefi (2008) presented a static distributed scheduling algorithm for scheduling parallel jobs in a computational grid. The global scheduling algorithm is responsible for allocating the submitted job to a cluster. Fuzzy logic is used to assign different weights for bandwidth, job communication requirements and job size. Gallet, Marchal and Vivien (2009) proposed a static scheduling approach to schedule workflow jobs in the Grid Environment with heterogeneous resources. The algorithm finds the optimal allocation by using a mixed linear programming approach.

*3.2.3.2 Dynamic Scheduling*

Dynamic scheduling allocates tasks to resources on the fly. This method is used when it is difficult to estimate job execution time and the job requests are arriving dynamically. Dynamic scheduling is more flexible since the placement of task is not fixed. Dynamic scheduling allows the reallocation of task to different resources during task execution. However, dynamic scheduling is more complex than static scheduling. If the dynamic scheduling is not properly designed, the scheduling may lead to system performance degradation and unnecessary job migrations. The dynamic scheduling algorithms are studied in (Viswanathan et al., 2007), (Lee and Zomaya, 2007), (Doulamis et al., 2007) and (Prodan and Wieczorek, 2010).

Viswanathan et al. (2007) proposed a Resource-Aware Dynamic Incremental Scheduling (RADIS) to handle large volumes of computationally intensive arbitrarily divisible loads. The algorithm considers dynamic arrival of loads with deadline constraints. A "pull-based" scheduling strategy with an admission control policy is implemented to satisfy the jobs' deadline requirements. Lee and Zomaya (2007) proposed two novel scheduling algorithms

namely the Shared-Input-data-based Listing (SIL) algorithm and the Multiple Queues with Duplication (MQD) algorithm for parameter sweep job. The algorithms make scheduling decisions without requiring the full accurate performance prediction information.

Doulamis et al. (2007) proposed a scheduling algorithm for fair scheduling. The algorithm uses a max-min fair sharing approach for providing fair access to users. When there is congestion, each user is given a certain weighting of the computation resources. Prodan and Wieczorek (2010) proposed a general bi-criteria scheduling heuristic called dynamic constraint algorithm (DCA). The algorithm consists of two phases that address the optimization problem of two independent criteria and is based on dynamic programming.

*3.2.3.3 Hybrid Scheduling*

Hybrid scheduling takes advantages of the static scheduling and dynamic scheduling. In hybrid scheduling, the scheduler will apply static scheduling when there is sufficient resources and task information. However, the placement of task is not fixed during execution. The scheduler will reallocate the task whenever there are changes in the resources performance. The hybrid scheduling algorithms are studied in (Korkhov et al., 2009) and (Zhu and Guo, 2009).

Korkhov et al. (2009) proposed an Adaptive Workload Load Balancing (AWLB) algorithm for parallel applications running on heterogeneous resources and User-Level Scheduling (ULS) environment. This algorithm minimizes the execution time of the parallel applications with divisible workload running on heterogeneous Grid resources. Zhu and Guo (2009) proposed a Hybrid Adaptive Genetic Algorithm (HAGA) to solve the

scheduling of dependent tasks in Grid. The algorithm improves the local search ability by adjusting the crossover and mutation probability adaptively and nonlinearly. Also, the algorithm improves the accuracy of convergence as well as the speed.

### 3.2.4 Objective Function

Each scheduler has its own objective function which is classified into application-centric and resource-centric (Zhu, 2003). Application-centric scheduler aims to optimize the performance of each individual application. Most of these schedulers try to minimize the makespan which is the time spent from the beginning of the first task in a job to the end of the last task of the job. These schedulers include (Dong and Selim, 2007), (Lee and Zomaya, 2007), (Gao et al., 2007), (Wu et al., 2007), (Rasooli et al., 2008), (Ding et al., 2009), (Falzon and Li, 2009) and (Machtans et al., 2009).

Besides makespan, the cost that an application needs to pay for resources utilization is another popular metric used by the scheduling algorithms. These schedulers include (Buyya et al., 2002), (Fard and Deldari, 2008), (Aoun and Gagnaire, 2009), (Arfa and Broeckhove, 2009), (Lu and Ma, 2009) and (Ranaldo and Zimeo, 2009). The cost function is an important metric for Cloud computing since the Cloud implements the pay-per-use model. Buyya et al. (2009) proposed a market-oriented resource pricing and allocation strategies to encompass customer-driven service management in Cloud environment.

Resource-centric scheduler aims to optimize the performance of the resources. The objective of these schedulers is to maximize resource utilization. Low utilization means a resource is going idle and wasted. Garg, Venugopal and Buyya, (2008) proposed a double

auction based meta-scheduler that uses valuation metrics to map user applications to resources. This scheduler provides load balancing across various resources. When the resources are fully utilized, both users as well as resource providers are able to gain the maximum benefits.

In addition, the resource-centric scheduler is very important in Cloud Computing since the service providers will be able to maximize their profits by maximizing the resources utilization. Table 3.1 shows the pricing of using Amazon Elastic Compute Cloud (Amazon EC2) with on demand instances. An efficient resource-centric scheduler allows the service providers to handle more requests and directly increases their profit.

**Table 3.1 Pricing of Amazon EC2 on demand instances (adapted from Amazon, 2010a)**

US – N. Virginia | US – N. California | EU – Ireland

| Standard On-Demand Instances | Linux/UNIX Usage | Windows Usage |
|---|---|---|
| Small (Default) | $0.085 per hour | $0.12 per hour |
| Large | $0.34 per hour | $0.48 per hour |
| Extra Large | $0.68 per hour | $0.96 per hour |
| **High-Memory On-Demand Instances** | **Linux/UNIX Usage** | **Windows Usage** |
| Extra Large | $0.50 per hour | $0.62 per hour |
| Double Extra Large | $1.20 per hour | $1.44 per hour |
| Quadruple Extra Large | $2.40 per hour | $2.88 per hour |
| **High-CPU On-Demand Instances** | **Linux/UNIX Usage** | **Windows Usage** |
| Medium | $0.17 per hour | $0.29 per hour |
| Extra Large | $0.68 per hour | $1.16 per hour |

In Cloud Computing, most of the applications are provided in Web Services. Thus, QoS requirements have become another metric in determining the resource allocation. The service providers need to aggregate the resources to provide an acceptable QoS to end users.

QoS is of particular concern to service providers as QoS directly affects end users' satisfaction and loyalty. Some of the active research studies of QoS in Grid and Cloud Computing include (Subrata et al., 2008), (Li et al., 2009a), (Li, 2009), (Sadhasivam et al., 2009) and (Xu et al., 2009). The details of QoS studies in Grid and Cloud Computing are discussed in chapter 5.

### 3.2.5 Gap Analysis

In summary, each scheduling algorithm has its strength and weaknesses. In order to have a better scheduling decision, the ability to determine the type of job requests and the resources available is essential. Furthermore, the major issues of job scheduling in Grid and Cloud Computing are mainly due to the heterogeneity and dynamic nature of the technologies.

In order to support a mixture of Grid and Cloud infrastructure, hybrid scheduling algorithm is used as the algorithm has the ability to implement static scheduling and dynamic scheduling. A Hybrid Scheduling Algorithm (HSA) which incorporates some the solutions to overcome the above issues, is proposed. HSA is reported to be able to maximize resources utilization and minimizes makespan in Grid and Cloud environment. In the next section, the proposed job scheduling algorithm is detailed.

## 3.3 Hybrid Scheduling Algorithm (HSA)

HSA contains a Cloud model, C, which consists of a number of sites that connected to one another through Wide Area Network (WAN).

$$C = \{S_1, S_2, \ldots, S_m\},\ 1 \leq i \leq m \text{ and } i, m \in \mathbb{N} \tag{3.1}$$

Each site, $S_i$, consists of different number of computational resources. $S_i$ is the $i^{th}$ site in the Cloud model. Each site is under the control of different administrative domains and is considered as different private Cloud in this model.

$$S_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,n}\},\ 1 \leq i \leq n \text{ and } i, n \in \mathbb{N} \tag{3.2}$$

The computation resources at each site are heterogeneous and consist of physical servers, compute clusters and desktop computers. The physical server is divided into application server and virtual server. The application server is preinstalled with an operating system and system software. The application server is able to execute a limited number of jobs depending on the pre-configured system software.

The virtual server provides a virtual environment that is used for creating a customized virtual machine according to the users' requests. The virtual server is able to execute all the jobs since the services are provided using an automatic deployment mechanism. The compute clusters consist of a group of computers that are connected to each other. Usually, the cluster consists of an administration front node and many cluster nodes. The administration front node handles the job requests and distributes the tasks to the cluster nodes. The desktop computer is the low-end personal computer used for daily activities by end users. The automatic deployment mechanism is required to deploy the job at each desktop computer.

Each resource may consist of different number of processors, processors of different speed or cores, amount of memory and storage space. At each site, there is a data repository which is connected to the computational resources through a high bandwidth Local Area Network (LAN). The data repository stores the operating system images, various types of application software, registry files and contains a Web Services repository. The Web Services repository consists of a number of Web Services package files and the Web Services provided by each site. The Web Services, $W_i$, is represented as below.

$$W_i = \{f_{i,1}, f_{i,2}, \ldots, f_{i,n}\},\ 1 \leq i \leq n \text{ and } i, n \in \mathbb{N} \tag{3.3}$$

The application model, A, consists of a number of independent jobs, $J_i$

$$A = \{J_1, J_2, \ldots, J_m\},\ 1 \leq i \leq m \text{ and } i, m \in \mathbb{N} \tag{3.4}$$

Each job model, $J_i$, consists of a number of n independent tasks. The tasks are submitted in the form of Web Services requests. Each request is equivalent to one single task. The tasks are classified as parametric tasks where all the tasks in the same job will execute the same Web Services with different parameter values. For example, a parametric task consists of two tasks with different parameter values such as Task(100) and Task(1000). Task(1000) will require a much longer time to execute as compared to Task(100). The execution time of the tasks is dependent on the parameter values in the parametric tasks.

$$J_i = \{T_{i,1}, T_{i,2}, \ldots, T_{i,n}\},\ 1 \leq i \leq n \text{ and } i, n \in \mathbb{N} \tag{3.5}$$

In this research, a HSA which maximizes the resource utilization and minimizes the total makespan is proposed. The hierarchical structure is implemented with the system having one centralized meta-scheduler and multiple distributed local schedulers. The meta-scheduler is responsible for accepting all the job requests and schedules the job request to each site based on the aggregate information collected from each site. The information collected from each site includes the number of computation resources available and a list of Web Services provided, $W_i$.

Firstly, the meta-scheduler considers all the sites that provide the Web Services as required by the job request. Then, the meta-scheduler will send the job requests to the site with the maximum number of available computation resources. The meta-scheduler does not use any prediction and the scheduling is performed in an on-line mode. The dynamic scheduling is applied because the meta-scheduler does not have the latest updated resources information for each individual site and by scheduling job on-the-fly, the scheduling process speeds up. To overcome the potential of a single point of failure in a centralized meta-scheduler, a backup meta-scheduler is implemented. Whenever a failure is detected in the primary meta-scheduler, a backup meta-scheduler with the same image and state as the primary meta-scheduler is created through virtualization. The image files as well as the registry are retrieved from the data repository. Figure 3.7 depicts the system architecture of the proposed scheduling algorithm.

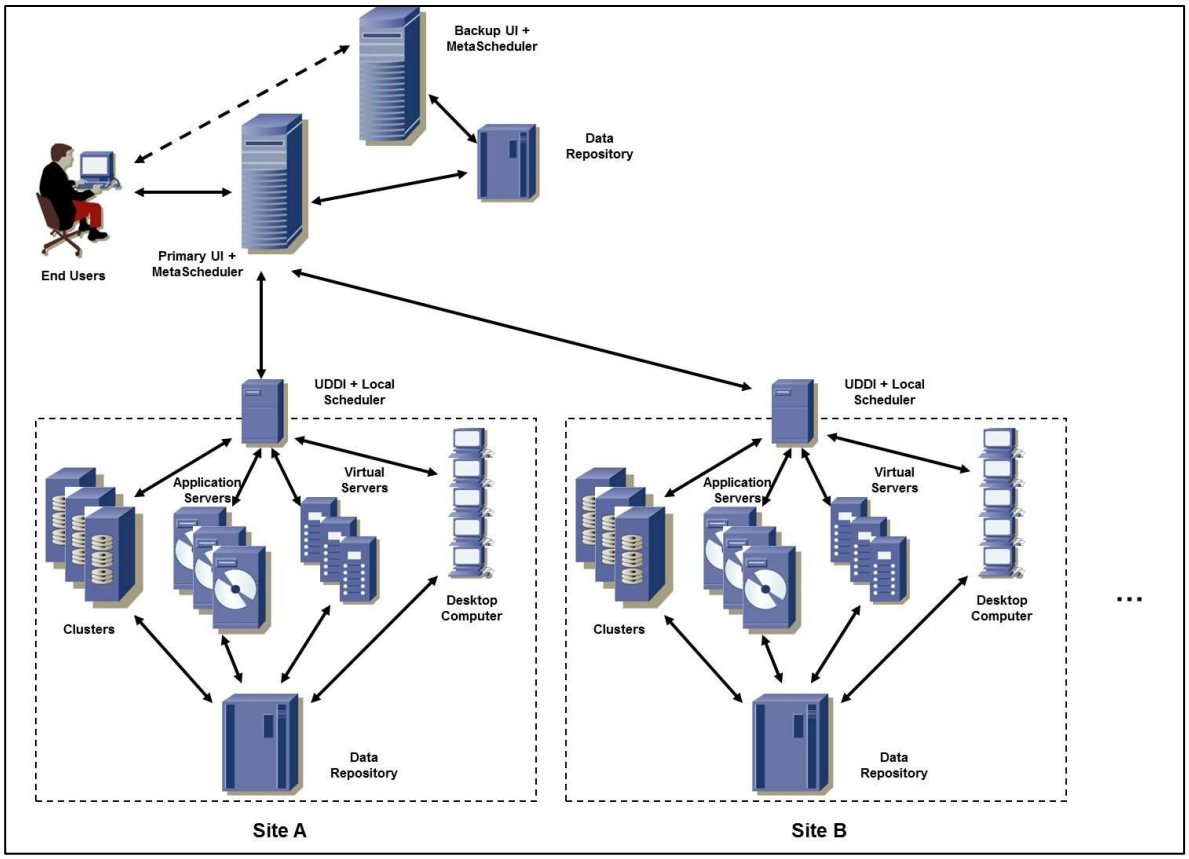


**Figure 3.7: System architecture of proposed algorithm**

As shown in figure 3.7, end users first submit job requests to a centralize User Interface (UI) server or meta-scheduler. The meta-scheduler then schedule the job requests to each site accordingly. Once the local scheduler receives the job request, the scheduler will perform static scheduling in making scheduling decision since the local scheduler has the updated resources information for its site. Firstly, the local scheduler sorts the unscheduled tasks for each job in descending order according to the parameter values. Since each job consists of several independent tasks that access the same Web Services, the parameter value used is gauged against the job length. A simple run is performed to determine the effect of incrementing or decrementing the parameter value on the job length, measured in terms of execution time. The sorted job requests list for $J_i$ is defined as

$$L_i = \{l_{i,1}, l_{i,2}, \ldots, l_{i,n}\},\ 1 \leq i \leq n \text{ and } i, n \in \mathbb{N} \quad (3.6)$$

where $l_{i,1}$ is the parameter value for each task $T_j$ in Job $J_i$

$$mean_i = \sum_1^n l_{i,j} / n, \text{ where } mean_i \text{ is the mean value for } L_i \quad (3.7)$$

Subsequently, the local scheduler generates a potential resource list for providing the Web Services. The potential resources are selected from the physical servers and the clusters but exclude the desktop computers since the desktop computers do not provide the Web Services. The detailed information such as number of processors (np), number of cores (nc) and processor speed (ps) are collected from each potential resource. These resources are classified as primary resource and are sorted. The sorted primary resource, $P_i$ for $J_i$, is defined as

$$P_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,n}\},\ 1 \leq i \leq n \text{ and } i, n \in \mathbb{N} \quad (3.8)$$

$$p_{i,j} = \{\ np(r_{i,j}), nc(r_{i,j}), ps(r_{i,j})\ \}$$

Besides the potential resources, the local scheduler also generates a desktop resources list. The desktop resources are derived from the entire available desktop computers. These resources are classified as secondary resource. The sorted secondary resource, $Q_i$ for $J_i$, is defined as

$$Q_i = \{q_{i,1}, q_{i,2}, \ldots, q_{i,n}\},\ 1 \leq i \leq n \text{ and } i, n \in \mathbb{N} \quad (3.9)$$

$$q_{i,j} = \{\ np(r_{i,j}), nc(r_{i,j}), ps(r_{i,j})\ \}$$

$$p_{i,j} \cap q_{i,j} = \emptyset$$

In order to allocate tasks to the resources in $Q_i$, an automatic deployment strategy is required. The details of the automatic deployment strategy are discussed in section 3.4.

Once $L_i$, $P_i$ and $Q_i$ are generated, the mean value, $mean_i$ is computed. This value is used to determine whether the task $T_{i,j}$ is assigned to the resources in $P_i$ or $Q_i$ based on the pseudo code below:

**if** $l_{i,j} > mean_i * z$ **then**
  allocate $T_{i,j}$ to $P_i$ that returns the minimum completion time
**else**
  allocate $T_{i,j}$ to $Q_i$ that returns the minimum completion time
**end if**

The task completion time for $T_{i,j}$ is given as $C_{i,j}$. The objective function of HSA is to maximize resource utilization and minimize makespan. The makespan is the maximum task completion time of all the completed tasks and is defined as,

$$M = maximum(C_{i,j}) \tag{3.10}$$

An optimum threshold value, z, is used to decide whether a task $T_{i,j}$ is assigned to the primary resources or secondary resources. Table 3.2 shows the results of the makespan (in seconds) of HSA running on different z values. Figure 3.8 shows the average makespan (in seconds) of HSA running on different z values.

**Table 3.2 Makespan (seconds) of HSA for random job arrival and random parameter values using different z values**

| Test | Job Description | z | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|---|
| 1 | Random job arrival and random parameter values (Group by Web Services) - HSA | 1.1 | 8027.20 | 8133.02 | 8038.32 | 8099.42 | 8175.71 | 8094.73 |
| 2 | Random job arrival and random parameter values (Group by Web Services) - HSA | 1.0 | 6775.57 | 6915.70 | 6886.68 | 6891.34 | 6890.57 | **6871.97** |
| 3 | Random job arrival and random parameter values (Group by Web Services) - HSA | 0.9 | 7371.88 | 7146.47 | 7285.13 | 7281.64 | 7169.59 | 7250.94 |
| 4 | Random job arrival and random parameter values (Group by Web Services) - HSA | 0.8 | 7766.88 | 7650.55 | 7562.51 | 7705.45 | 7571.69 | 7651.42 |
| 5 | Random job arrival and random parameter values (Group by Web Services) - HSA | 0.7 | 8122.32 | 8156.73 | 8111.51 | 8166.72 | 8131.97 | 8137.85 |

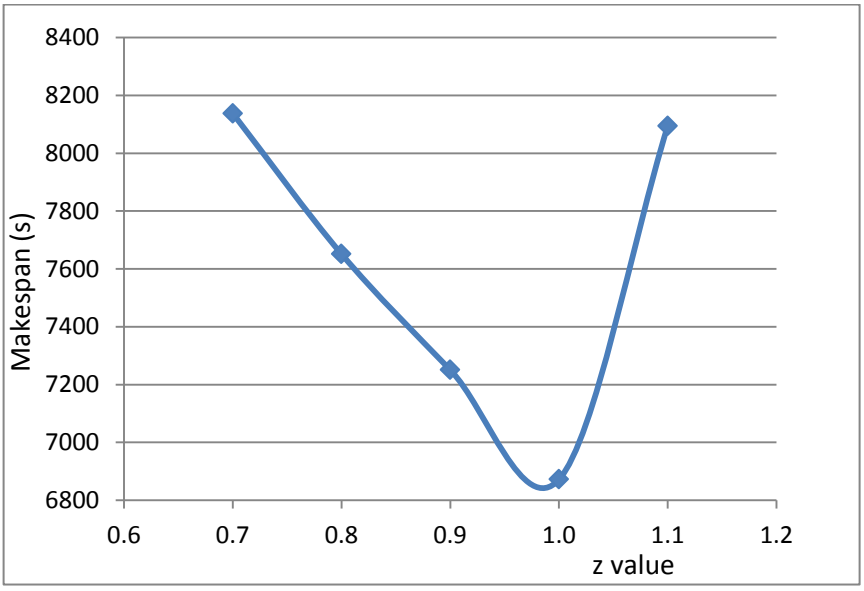


**Figure 3.8: Makespan (seconds) of HSA for random job arrival and random parameter values using different z values**

Table 3.2 shows that HSA returns the minimum makespan when the z value is 1.0. This is further depicted in Figure 3.8. This is the optimum threshold value used for the experiment. The z value is identified by running the same set of experiment at various times with an increment or decrement of 0.1. The optimal z value is the threshold value that generates a schedule list with the shortest makespan. Figure 3.9 depicts the proposed HSA scheduling algorithm.

Input:
    $J_i$, the job request that consists of n independent task $T_{i,j}$
    $p_m$ or $q_n \in S_i$, , where $S_i$ is a set of resources in site i
Output
    A schedule of $T_{i,j}$ into $p_m$ or $q_n$

// group the tasks according to Web Services at each interval
∀ Job request, $J_i$, in waiting queue
  Sort the task $T_{i,j}$ in descending order based on parameter value
  Generate a sorted job requests list for $J_i$, $L_i$
  Compute $mean_i$ from $L_i$

  ∀ servers and clusters in site I  // generate the primary resource list
    Select potential resources and generate sorted primary resource list, $P_i$

  ∀ desktop computers in site I  // generate the secondary resource list
    Select available desktop computers and generate sorted secondary resource list, $Q_i$

  // compare the multiplication of z and mean with the parameter value to
  // determine whether to schedule the task to primary or secondary resource
  ∀ Task, $T_{i,j}$
    if $l_{i,j} > mean_i * z$
      allocate $T_{i,j}$ to $p_m$ that returns the minimum completion time
    Otherwise
      allocate $T_{i,j}$ to $q_n$ that returns the minimum completion time
    Repeat the experiment with different z value, with $z \pm 0.1$
    to get the minimum makespan

**Figure 3.9: Proposed HSA**

### 3.4 Automatic Deployment Mechanism

Figure 3.10 shows the main components of the HSA and automatic deployment. As discussed in previous section, the UI provides an interface to support job submission. The meta-scheduler performs scheduling of jobs to multiple sites while the local scheduler schedules the tasks according to each individual resource at each site. Besides, the resource manager provides all the resource information for each site. Once the local scheduler has the job information and resource information, the scheduler generates a job-to-resource mapping list. Finally, the deployer dispatches the tasks to each individual resource for execution.

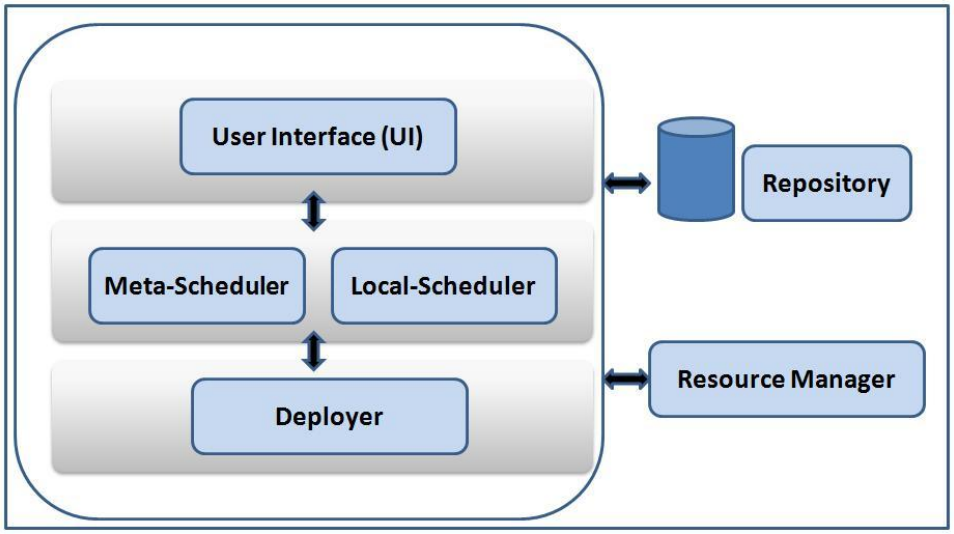


**Figure 3.10: Main components of HSA and automatic deployment**

Since the resources in Cloud environment are heterogeneous, different deployment methods are required to deploy the tasks to each individual resource. In order to execute the Web Services running on the physical server with preinstalled operating system and Web server, a direct invocation of the Web Services is required. Since the execution of the

services required a significant amount of time to process, the asynchronous communication invocation is used to allow other threads to continue execution without blocking.

There are two stages involve in order to execute the Web Services running on the virtual server. The first stage is to create a virtual infrastructure. A virtual infrastructure is configured to allow the sharing of multiple virtual servers at each site. Each virtual infrastructure contains a resource manager that decides the number of virtual server to be created with the required software applications and platforms. Once the virtual infrastructure is created, the second stage is to create the virtual server. At this stage, the required image files as well as the Web Services are gathered and configured to generate the relevant virtual server according to the job request. The Web Services are then invoked using asynchronous communication.

In order to execute the Web Services running on Clusters, the cluster plug-in must be installed on the Web server and running on the administrator node. Once the cluster is created, the instances are created for each specific cluster nodes. In this research, the Glassfish Web server is installed on Rocks cluster.

As discussed in previous section, the automatic services deployment mechanism is used to deploy the Web Services to the secondary resource which consists of desktop computers. The automatic services deployment uses two approaches to deploy the Web Services running on desktop computer. The first approach deploys the services to the .NET framework resources. The deployer first retrieves the service *.asmx* file from the Web Services repository and deploys the service on the resource. An agent in the resource

creates a Web Services end point with IP address and port number that map to the *.asmx* file. Http.sys architecture is used as the low-level HTTP protocol stacks for the service deployment. Http.sys architecture is a kernel-mode component that offers HTTP services to all applications on the computer. When http.sys receives a request, http.sys can forward it directly to the correct work process. Http.sys is capable of caching responses directly within the kernel. This improves the overall throughput and performance. Once the Web Services are deployed the asynchronous communication invocation is used to execute the Web Services. Figure 3.11 shows the Http.sys architecture.

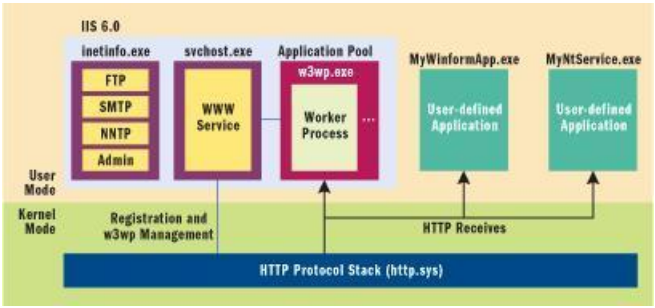


**Figure 3.11: Http.sys architecture**

The second approach is implemented on the Java 2 Platform Enterprise Edition (J2EE) environment. In this approach, the manager first retrieves the JAR file from the Web Services repository and deploys the service on the resource. The JAX-WS 2.0 Endpoint Class is used to create the Web Services end point. The JAR file contains the Web Services descriptions and implementation. The Web Services bind to an end point with a WSDL address. Since the execution of the services requires a significant amount of time to process, the asynchronous communication invocation is again used to invoke the services. Once the job is completed, the output is stored in the UI server and the user is notified.

The automatic Web Services deployment architecture is implemented on .NET and J2EE environment. Since the Web Services resolve the interoperability problem, Web Services can be deployed to run on different platforms.

### 3.5 Experiment Setup

A testbed which implements the above components in Grid and Cloud environment is deployed to carry out testing and evaluations of the proposed algorithm. The testbed allows the registration of different types of Web Services as well as new resources. Parametric type of Web Services is selected for the experiment, in which the execution times of these services are affected by the parameter values. In addition, most of these Web Services consists of the Monte Carlo implementation. The Monte Carlo method generates random sampling and is compute intensive when the sample size is huge. Furthermore, the testbed also supports the implementation of the MIN-MIN and MAX-MIN heuristics scheduling algorithm. The test case of the implementation correctness of the proposed HSA and automatic deployment algorithm, MIN-MIN and MAX-MIN is described in Appendix A.1.

To evaluate the performance of the proposed algorithm, a predefined set of resource and job parameters used is shown in Table 3.3. A set of comparisons, in similar settings as Table 3.3 is performed to evaluate the proposed algorithm as compared to MIN-MIN and MAX-MIN scheduling algorithm. The performance metric used for the comparison is makespan. Several test cases which include 1) Random job arrival with random job parameter values; 2) Random job arrival with big gap job parameter values; 3) Random job arrival with Poisson distribution parameter values are conducted. The time interval use for

the experiment is twenty seconds per interval for local scheduler. The size of the job length is based on the parameter values. For the propose verifying the scheduling algorithm, the experiments are conducted using different fixed value of z for the same test case.

**Table 3.3 A predefined set of resources and job parameters**

| Description | Value Range |
|---|---|
| Number of sites | 3 – 5 |
| Number of hosts per site | 20 – 50 |
| Number of CPU | 1 – 4 |
| Number of core per CPU | 1 – 8 |
| CPU speed | 0.8 – 3.2 GHz |
| Number of tasks | 300 – 500 |
| Execution time of task (second) | 1 – 1000 |
| Type of Web Services | 1 – 10 |

## 3.6 Results and Analysis

Table 3.4 – 3.6 show the results of the makespan (in seconds) for different test cases using the three different algorithms.

**Table 3.4 Makespan (seconds) of different algorithms for random job arrival and random parameter values**

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Random job arrival and random parameter values (Group by Web Services) - MIN-MIN | 7302.86 | 7102.39 | 7196.85 | 7335.69 | 7292.59 | 7246.08 |
| 2 | Random job arrival and random parameter values (Group by Web Services) - MAX-MIN | 6851.26 | 6931.62 | 6974.93 | 6946.60 | 6938.74 | 6928.63 |
| 3 | Random job arrival and random parameter values (Group by Web Services) - HSA (z=1.1) | 8027.20 | 8133.02 | 8038.32 | 8099.42 | 8175.71 | 8094.73 |
| 4 | Random job arrival and random parameter values (Group by Web Services) - HSA (z=1) | 6775.57 | 6915.70 | 6886.68 | 6891.34 | 6890.57 | **6871.97** |
| 5 | Random job arrival and random parameter values (Group by Web Services) - HSA (z=0.9) | 7371.88 | 7146.47 | 7285.13 | 7281.64 | 7169.59 | 7250.94 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | Random job arrival and random parameter values (Group by Web Services) - HSA (z=0.8) | 7766.88 | 7650.55 | 7562.51 | 7705.45 | 7571.69 | 7651.42 |

**Table 3.5 Makespan (seconds) of different algorithms for random job arrival and big gap parameter values**

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Random job arrival and big gap parameter values (Group by Web Services) - MIN-MIN | 10142.81 | 10100.90 | 10233.82 | 10187.81 | 10393.51 | 10211.77 |
| 2 | Random job arrival and big gap parameter values (Group by Web Services) - MAX-MIN | 9210.69 | 9160.25 | 9266.51 | 9100.62 | 9322.25 | 9212.06 |
| 3 | Random job arrival and big gap parameter values (Group by Web Services) - HSA (z=1.7) | 11357.10 | 11441.58 | 11587.69 | 11414.98 | 11263.90 | 11413.05 |
| 4 | Random job arrival and big gap parameter values (Group by Web Services) - HSA (z=1.6) | 9080.02 | 9086.91 | 9098.26 | 8988.36 | 8972.82 | **9045.27** |
| 5 | Random job arrival and big gap parameter values (Group by Web Services) - HSA (z=1.5) | 9899.65 | 9873.44 | 9651.76 | 9892.85 | 9905.43 | 9844.63 |
| 6 | Random job arrival and big gap parameter values (Group by Web Services) - HSA (z=1) | 11251.15 | 11286.79 | 11164.52 | 11290.21 | 11297.55 | 11258.04 |

**Table 3.6 Makespan (seconds) of different algorithms for random job arrival and Poisson distribution parameter values**

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Random job arrival and Poisson distribution parameter values (Group by Web Services) - MIN-MIN | 5680.52 | 5683.58 | 5626.12 | 5637.25 | 5637.57 | 5653.01 |
| 2 | Random job arrival and Poisson distribution parameter values (Group by Web Services) - MAX-MIN | 5151.47 | 5153.45 | 5112.64 | 5000.93 | 5019.27 | 5087.55 |
| 3 | Random job arrival and Poisson distribution parameter values (Group by Web Services) - HSA (z=1) | 6256.13 | 6321.85 | 6280.41 | 6278.34 | 6251.25 | 6277.60 |
| 4 | Random job arrival and Poisson distribution parameter values (Group by Web Services) - HSA (z=0.8) | 5039.44 | 5095.96 | 5089.63 | 5035.08 | 5016.86 | **5055.39** |
| 5 | Random job arrival and Poisson distribution parameter values (Group by Web Services) - HSA (z=0.7) | 6149.34 | 6178.30 | 6130.86 | 6144.84 | 6183.42 | 6157.35 |
| 6 | Random job arrival and Poisson distribution parameter values (Group by Web Services) – HSA (z=0.5) | 6422.74 | 6461.84 | 6437.99 | 6366.85 | 6395.84 | 6417.05 |

Table 3.4 shows that when the z value is 1, HSA outperforms MIN-MIN and MAX-MIN scheduling algorithms. However when the z value is 1.1, the average makespan value is higher as compared to MIN-MIN and MAX-MIN scheduling algorithms. The results also showed that that the minimum average makespan is achieved when the z value is 1, as compared to z value is 0.9 or 1.1.

Table 3.5 shows that when the big gap parameter values are used for the experiment, HSA outperforms MIN-MIN and MAX-MIN scheduling algorithms with the best timing obtained when the z value is 1.6. The z values in this experiment are larger than the values used for the random parameter values. The proposed HSA yielded a larger makespan over MIN-MIN and MAX-MIN scheduling algorithms when the z value is equal to 1 or 1.7.

Table 3.6 shows that for the random arrival tasks with Poisson distribution parameter values, HSA achieves better result over MIN-MIN and MAX-MIN when the z value is equal to 0.8 but worse for other z values. This indicates that the z value is very important in determining the effectiveness of HSA.

Figure 3.12 shows the makespan (in seconds) for different experimental cases using the three different algorithms. Table 3.7 tabulates the results of the makespan (in seconds) and the percentage of improvements for the three different algorithms. It is observed that HSA has made between 5% - 11% improvements over the MIN-MIN scheduling algorithm while making only shows a slightly improvement over MAX-MIN scheduling algorithm.

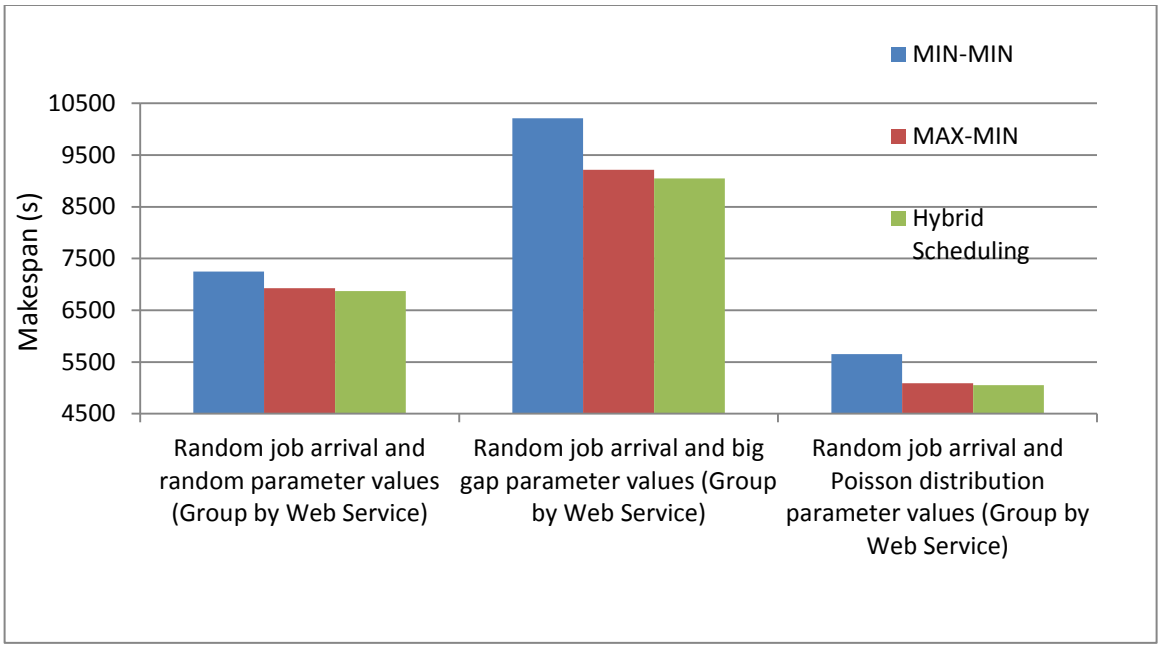


**Figure 3.12: Makespan (seconds) of different algorithms for different experimental cases**

**Table 3.7 Makespan (seconds) of different algorithms for different experimental cases and percentage of improvement**

| Test | Job Description | MIN-MIN | MAX-MIN | HSA | z | % improvement of HSA as compared to MIN-MIN | % improvement of HSA as compared to MAX-MIN |
|---|---|---|---|---|---|---|---|
| 1 | Random job arrival with random parameter values (Group by Web Services) | 7246.08 | 6928.63 | 6871.97 | 1.0 | 5.2 | 0.8 |
| 2 | Random job arrival and big gap parameter values (Group by Web Services) | 10211.77 | 9212.06 | 9045.27 | 1.6 | 11.4 | 1.8 |
| 3 | Random job arrival and poisson distribution parameter values (Group by Web Services) | 5653.01 | 5087.55 | 5055.39 | 0.8 | 10.6 | 0.6 |

According to the experimental results, the superior performance of HSA is achieved through the proper selection of the mean value at each interval and the z value that decides

whether to schedule tasks to the primary resources or secondary resources. The ability to adjust the mean value at each interval provides more effective scheduling decisions due to the heterogeneity and dynamic nature of Grid and Cloud Computing.

In MIN-MIN and MAX-MIN scheduling algorithms, all the resources are considered as primary resource and the Web Services must be preconfigured. In contrast, HSA classifies the resources into primary resources and secondary resources. With the implementation of automatic deployment mechanism, HSA is able to automate the Web Services deployment process to any type of resources. This provides more flexibility as compared to MIN-MIN and MAX-MIN scheduling algorithms.

### 3.7 Chapter Summary

This chapter presented the solution that used to address the job scheduling problem in Grid and Cloud environment. A Hybrid Scheduling Algorithm (HSA) with automatic deployment mechanism is proposed to maximize resources utilization and minimizes makespan. In HSA, a meta-scheduler is responsible for accepting all the job requests and scheduling the job requests to multiple sites. The meta-scheduler does not use any prediction information and the scheduling is performed in an on-line mode. When the local scheduler receives the job request, the scheduler performs static scheduling in making scheduling decision. Parametric type of Web Services is used for the experiment, in which the execution time of these services is dependent on the parameter values. Besides, the resources are classified into primary resources and secondary resources. The primary resources consist of servers and the clusters while the secondary resources consist of

desktop computers. The local scheduler performs the scheduling decision based on the job and resource information with the objective of minimizing makespan.

In addition, an automatic deployment mechanism is proposed to automate the process of Web Services deployment to the secondary resources. Automatic deployment mechanism allows the Web Services to be deployed dynamically to the resources without the needs of Web server. Furthermore, this mechanism is a light weight approach that is usable for deploying any type of Web Services to the .NET framework resources or J2EE environment. The proposed algorithm is then compared to the MIN-MIN and MAX-MIN scheduling algorithms. Experimental results show that HSA and automatic deployment mechanism minimize the makespan by 1% – 11%.

The advantages of using HSA and automatic deployment mechanism are that the algorithm supports new types of Web Services. HSA is suitable when there are insufficient historical records to perform benchmarking. Besides, the automatic mechanism allows the Web Services to be executed on the desktop computers. However, the drawback of this algorithm is that all the Web Services have to be grouped at each interval before making scheduling decision. This is because different Web Services have different parameter values and the mean value is not meaningful when computed using different Web Services. Another drawback of HSA is that different experiments have to be conducted for similar settings to get the optimal z value to minimize the makespan. This is because different experimental sets require different z values.

In the next chapter, an enhancement of HSA, Adaptive Scheduling Algorithm (ASA) is proposed. ASA dynamically determines the z value at each interval. In addition, the application and resource benchmarking is introduced to address the issue of Web Services grouping.

# CHAPTER 4 ADAPTIVE SCHEDULING ALGORITHM (ASA)

This chapter presents the enhancement of HSA using benchmarking and adaptive mechanism. The first section describes about the job length estimation using application benchmarking. This is followed by the discussion of resources benchmarking. The third section proposes the enhancement of HSA which is ASA. The final section describes the experiment setup and evaluates the performance of ASA.

## 4.1 Application Benchmarking

Computational Grid comprises a large number of heterogeneous resources located at different administrative domains. Each resource has a variety of capabilities and supports different type of applications. One of the key challenges in Grid environment is to select adequate resources to satisfy the job requirements. To address the above challenge, adequate resource information and job attributes are required. The accuracy of these information guarantees that the job submitted is deployed to the most appropriate resources. Benchmarking is one of the methods that is used to extract the job attributes and the performance of a resource for different type of jobs.

Benchmarks are standardized programs or detailed specifications of programs designed to investigate well-defined performance properties of computer systems according to a widely accepted set of methods and procedures (Weicker, 2002). Benchmarking has played an important role in conducting the performance analysis of optimization algorithms in Grid and Cloud Computing. Benchmarking is used to compare the performance of the algorithm

for well-known NP-hard optimization problems. In addition, benchmarking is used as a means for the evaluation of scheduling algorithm.

Grid benchmarking is defined as the use of benchmark programs for the fair, concise, and affordable performance characterization of different aspects of a Grid infrastructure (Dikaiakos, 2007). Grid architecture can benefit substantially from using standard means for measurement and for comparison of design alternatives. Dikaiakos (2007) and Snavely et al. (2003) presented the importance of benchmarking in the evaluation of the computational resources in Grid environment.

A Grid benchmark is generated by combining three characteristics; namely, computing consistency, heterogeneity of resources and heterogeneity of jobs (Fatos et al., 2009). The computing consistency measures the performance of the resources in handling different type of jobs. A resource is consistent when the resource runs faster than other resources for all the jobs. Inconsistency means that a resource is faster for some jobs and slower for some others. The heterogeneity of resources and jobs represents the actual Grid and Cloud environment.

With these characteristics, benchmarking can assess the performance of Grid and Cloud by using different parameters such as instance size, type of jobs and type of resources. The instance size specifies the number of instance and the size of each instance to run for benchmarking. Also, different types of jobs are used to generate different workloads to evaluate the scheduling algorithms for different Grid and Cloud environment. In addition,

different type of resources with different capabilities, platforms and software are used in the benchmark evaluations.

### 4.1.1 Related Works

With the widespread use of the Grid systems, a number of research groups are focusing on different aspects of Grid benchmarking, proposing benchmark specifications, benchmarking suites, and benchmarking tools for Grid. NAS Grid Benchmark (NGB) is one of the first proposed Grid benchmark. NGB is designed to serve as a uniform tool for testing the functionalities and efficiencies of the Grid environment. The NGB specifications define a set of computationally intensive, synthetic Grid benchmarks. The job turnaround time is proposed as the performance metric for the benchmarking (Frumkin and Van der Wijngaart, 2001).

As the Grid environment is rapidly expanding in size and complexity, the task of benchmarking and testing becomes more unmanageable. Tsouloupas and Dikaiakos (2007) proposed an integrated tool named GridBench to facilitate Grid benchmarking. GridBench supports the testing, benchmarking and ranking of Grid resources. In addition, GridBench supports the definition, deployment and execution of parameterized tests and benchmarks on the Grid. At the same time, GridBench allows the validation, archival, retrieval, and analysis of test results.

Fatos et al. (2009) proposed a static benchmarking for Grid scheduling. The benchmarking is generated using a discrete event-based Grid simulator, namely HyperSim-G, with different instance sizes, type of jobs and type of resources. The objective of this benchmark

for Grid scheduling is to reveal the performance of algorithms and the heuristic methods utilised. The static benchmark is useful because the experimental study can be repeated as many times as needed to obtain relevant statistical results.

Clematis et al. (2010) proposed a two-level benchmarking methodology to facilitate the ranking of Grid resources and the submission of jobs. The Grid benchmark is divided into two categories which are micro-benchmark and application-specific benchmark. The micro-benchmark profiles resources by computing the number of floating point operations per second to represent CPU performance, the latency and the bandwidth to evaluate the interconnection performance. The application-specific benchmark measures the performance of the resources with different type of job behaviours. This benchmark is able to analyse the behaviour of resources for a class of jobs and is very useful for frequently used applications.

Besides the research studies on benchmarking on resources and jobs, some research groups are focused on the benchmarking the Grid services. As Grid move towards SOA, the performance of Grid such as resource management and job monitoring is getting critical and can affect the overall quality of a Grid environment. A badly designed Grid Service may lead to low availability and adding overhead to the job makespan. Furthermore, new benchmarks are required to measure Grid-service dependability. Grid-service benchmarks are important tools for assessing the performance of Grid services and comparing different service configurations. Plale et al. (2004) proposed a micro-kernel benchmark to measure the response time and throughput of GIS.

In summary, the application benchmark is able to provide the performance of the resources running on different type of jobs. Since the scheduling decision depends largely on the accuracy of this performance information, application benchmark is a very important process to aid the scheduling decision.

### 4.1.2 Implementation of Application Benchmarking

In relation to the above, this research uses the application-specific benchmark to measure the performance of resources with different type of jobs. This approach is implemented because most of the Web Services is frequently executed in a private Cloud environment. The application-specific benchmark is used to estimate the job length for different Web Services running on different resources. In chapter 3, the parameter value is used as the job length for each job request. Although the parameter value is used to determine the job length for the same Web Services, it is unable to differentiate the job length for different Web Services. To address the above problem, benchmarking is used to estimate the job length. The benchmarking is implemented using the Cloud testbed and running on combinations of different resources and jobs. The testbed is used as the empirical study of various parameter values.

With application-specific benchmark, the job length is estimated by running the same Web Services with different parameter values on the same resources. Then, curve fitting is used to construct a mathematical function that has the best fit from the historical records. The curve may be linear, exponential or polynomial. The mathematical function is then used to estimate the job length of the same Web Services with different parameter values. The characteristics of the Web Services are evaluated through application benchmarking.

In order to estimate the job length running on different resources, the same set of Web Services with same parameter value are used on different set of resources. The parameter value is used as the job length in the experiment. When the value is bigger, the execution time will be longer. The CPU speed is used as the metric to differentiate the resources performance since the Web Services used in the experiments are compute intensive. Then, a ratio is derived using the historical records and the job length is estimated when the Web Services are executed on the resources with different CPU speed. The assumption of the application-specific benchmark used in this research is that the performance of the resources is largely dependent on the CPU.

Table 4.1 – 4.3 show the benchmark results for job execution time when running the same type of Web Services with different parameter values on resource A, resource B and resource C. The parameter value is used to identify the job length. The higher value of the parameter value means that the length of the job is longer and require more time to execute.

Resource A is a workstation with Intel Core2 Duo E6550 and CPU speed 2.33 GHz. Resource B is a server with Intel Xeon E5320 and CPU speed 1.86 GHz. Resource B consists of two CPUs with quad cores for each CPU. The resource is running on .NET framework. Resource C is the cluster with two child nodes. Each child node is running on Intel Xeon W3520 and CPU speed 2.67GHz. Resource A and Resource B are running on .NET framework while Resource C is running on J2EE environment. Five experiments will be conducted for each test.

**Table 4.1 Benchmark results: average execution time of service 1 running on resource A**

| Test | Job Parameter Value | Execution Time | | | | | Average Execution Time |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | |
| 1 | 500000 | 2.1 | 2.1 | 2.2 | 2.0 | 2.0 | 2.08 |
| 2 | 1000000 | 4.1 | 4.1 | 4.2 | 3.9 | 4.0 | 4.06 |
| 3 | 3000000 | 6.9 | 7.0 | 7.2 | 7.1 | 7.0 | 7.04 |
| 4 | 5000000 | 18.3 | 18.5 | 18.3 | 18.3 | 18.2 | 18.32 |
| 5 | 10000000 | 36.8 | 36.5 | 36.4 | 36.4 | 36.7 | 36.56 |
| 6 | 30000000 | 102.5 | 101.8 | 102.6 | 102.3 | 102.7 | 102.38 |
| 7 | 50000000 | 181.0 | 181.5 | 181.1 | 180.4 | 180.9 | 180.98 |
| 8 | 100000000 | 356.9 | 359.7 | 359.1 | 362.4 | 357.1 | 359.04 |
| 9 | 300000000 | 1046.2 | 1046.9 | 1044.7 | 1045.2 | 1047.8 | 1046.16 |
| 10 | 500000000 | 1804.0 | 1810.5 | 1806.7 | 1805.2 | 1808.6 | 1807.00 |

**Table 4.2 Benchmark results: average execution time of service 1 running on resource B**

| Test | Job Parameter Value | Execution Time | | | | | Average Execution Time |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | |
| 1 | 500000 | 2.1 | 2.1 | 2.0 | 2.0 | 2.0 | 2.04 |
| 2 | 1000000 | 2.3 | 2.4 | 2.2 | 2.5 | 2.3 | 2.34 |
| 3 | 3000000 | 6.8 | 6.8 | 6.7 | 6.9 | 6.9 | 6.82 |
| 4 | 5000000 | 9.2 | 9.4 | 9.2 | 9.3 | 9.3 | 9.28 |
| 5 | 10000000 | 19.1 | 18.9 | 18.8 | 19.2 | 18.9 | 18.98 |
| 6 | 30000000 | 41.6 | 41.3 | 41.5 | 41.6 | 41.5 | 41.50 |
| 7 | 50000000 | 72.5 | 73.1 | 73.2 | 73.6 | 73.9 | 73.26 |
| 8 | 100000000 | 152.9 | 153.4 | 152.4 | 152.6 | 152.7 | 152.80 |
| 9 | 300000000 | 412.6 | 413.3 | 413.5 | 412.7 | 412.9 | 413.00 |
| 10 | 500000000 | 720.0 | 731.0 | 732.3 | 725.0 | 727.0 | 727.06 |

**Table 4.3 Benchmark results: average execution time of service 1 running on resource C**

| Test | Job Parameter Value | Execution Time | | | | | Average Execution Time |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | |
| 2 | 1000000 | 6.8 | 6.8 | 6.7 | 6.8 | 6.8 | 6.78 |
| 3 | 3000000 | 7.5 | 7.6 | 7.5 | 7.6 | 7.5 | 7.54 |
| 4 | 5000000 | 9.9 | 9.7 | 9.8 | 9.8 | 9.8 | 9.80 |
| 5 | 10000000 | 20.6 | 21.3 | 20.3 | 20.7 | 21.1 | 20.80 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 30000000 | 56.2 | 54.1 | 52.9 | 53.8 | 54.8 | 54.36 |
| 7 | 50000000 | 93.7 | 92.8 | 93.2 | 93.4 | 93.5 | 93.32 |
| 8 | 100000000 | 190.2 | 184.4 | 189.8 | 192.7 | 194.2 | 190.26 |
| 9 | 300000000 | 512.4 | 507.6 | 508.6 | 509.3 | 510.3 | 509.64 |
| 10 | 500000000 | 919.7 | 925.6 | 932.2 | 923.7 | 928.4 | 925.92 |

Figure 4.1 – 4.3 depict the relationship between the parameter value and the makespan. From the figures, both the parameter value and makespan are linearly proportional to one another. A linear trend line is added to the graph and the mathematic equation is formed. From the equation, the execution time for the same Web Services with different parameter values is estimated. The estimation of the execution time gets more accurate when more historical records are made available.

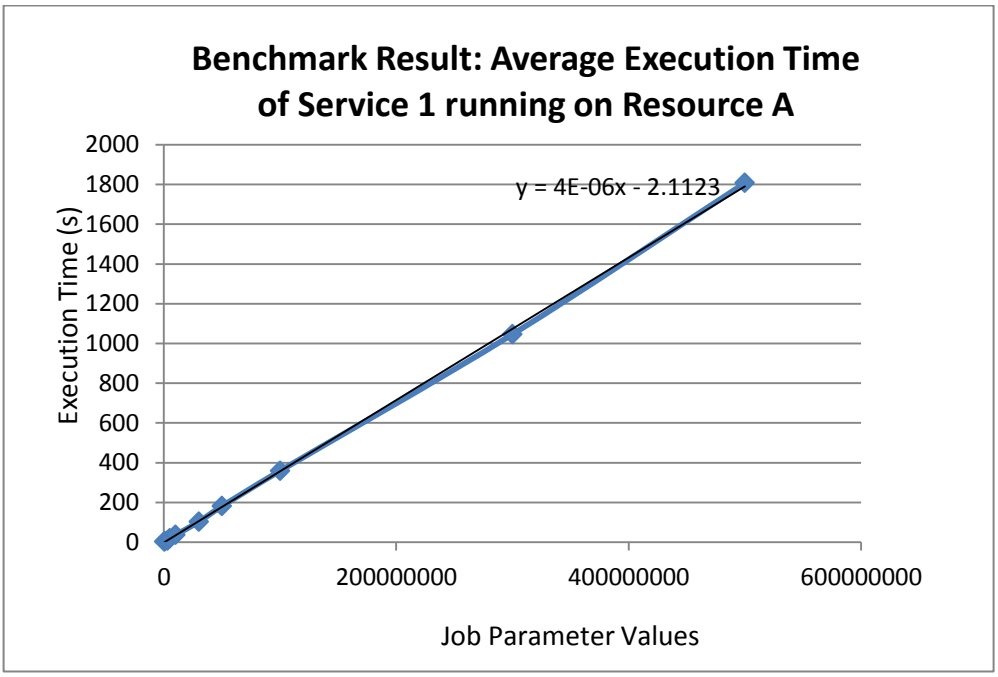


**Figure 4.1 Benchmark result: average execution time of service 1 running on resource A**

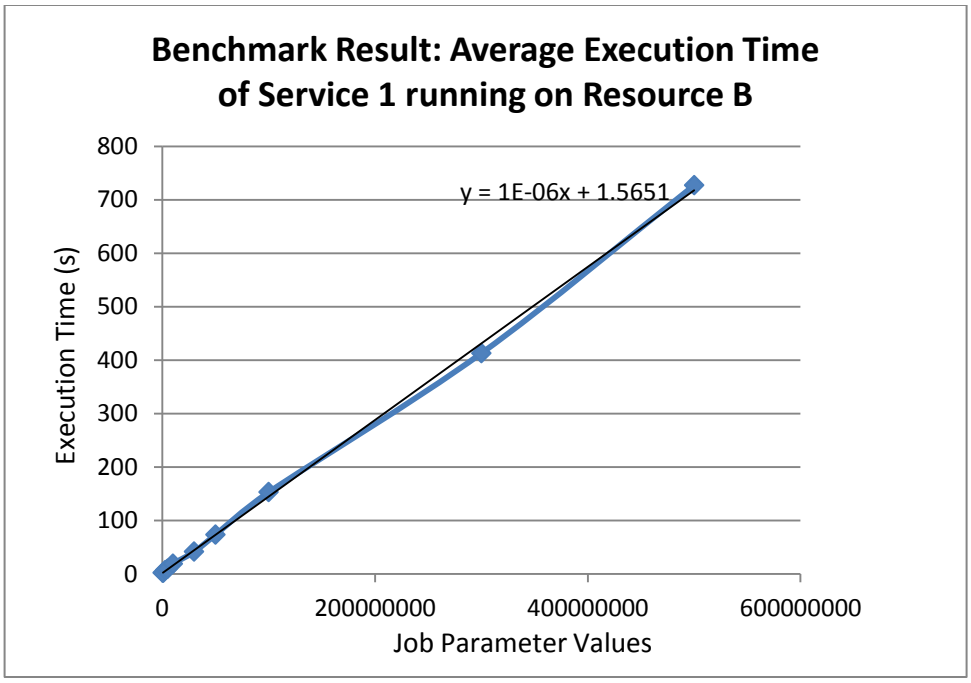


**Figure 4.2 Benchmark result: average execution time of service 1 running on resource B**

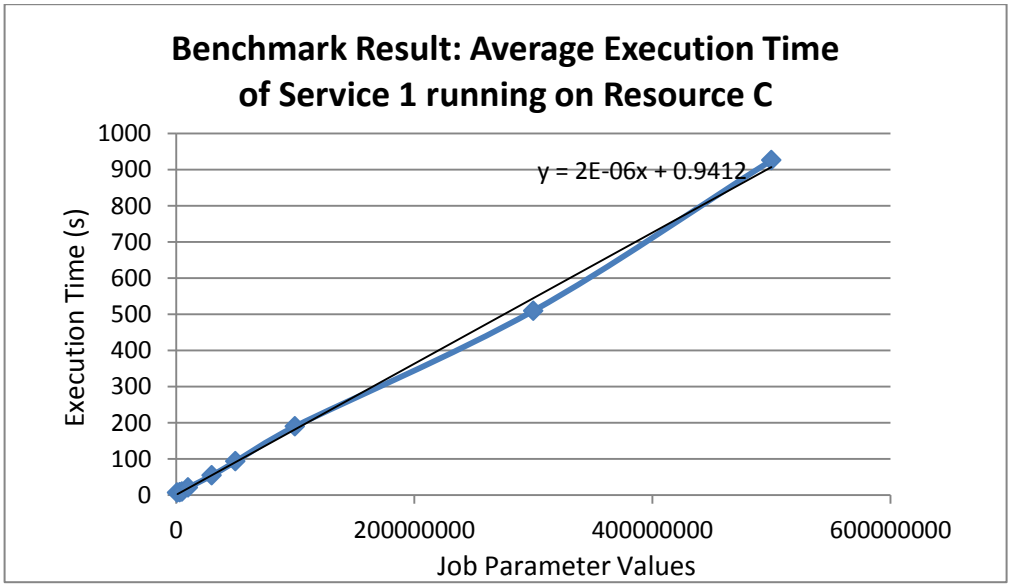


**Figure 4.3 Benchmark result: average execution time of service 1 running on resource C**

## 4.2 Resource Benchmarking

Nowadays, computers are getting powerful and with most of them having multiple CPUs and CPUs with multiples cores. At the same time, the applications are becoming more complex and places greater demand on the system performance. As such, the CPU, memory, cache and IO play important roles in determining the execution time of the application. Thus, there is a need to understand the performance differences of the above requirements running on different resources.

There are many studies being done on the general purpose processors such as those from Intel and AMD, to compare the resources performances on different computer architectures. For this purpose, the industries and academies have developed many different benchmark suites. Some of them are: PerformanceTest 7.0 (PassMark, 2010), CPU2006 (SPEC, 2010) and PCMark05 (FutureMark, 2010). Most of these benchmark suites comprised of a number of application programs and each of the benchmark consists of hundreds of billions of dynamic instructions. Besides, each benchmark suite uses a different mechanism to measure the performance of the CPU, memory, graphics memory, hard disk and system.

### 4.2.1 Related Works

PerformanceTest 7.0 (2010) is created by PassMark to benchmark the computer systems using a variety of different speed tests. The result of the benchmark is expressed as a rating that is used to compare the performance of different computer systems. The higher the rating, the better the computer system performs. Various performance benchmarking are conducted using PerformanceTest 7.0. These include CPU test, graphic test, disk I/O test and memory test. The most important test in PerformanceTest 7.0 is the CPU test.

PerformanceTest 7.0 performs one simultaneous CPU test for every logical CPU, physical CPU core or physical CPU package. In order to get the CPU benchmark rating, eight different tests that includes mathematical operation test, floating point number test, prime number test, data compression and encryption test, image rotation test and string sorting test are carried out. In addition, the benchmarking also includes the Streaming Single Instruction Multiple Data Extensions (SSE) and 3DNow test. Both tests allow the 128-bit and 64-bit floating point mathematical and logical operations. PerformanceTest 7.0 is used by (Martinovic et al., 2010) to measure the performance of the virtual machine running on different operating systems.

CPU2006 (2010) is a single-threaded compute-intensive benchmark developed by Standard Performance Evaluation Corporation (SPEC). This benchmark suite is used by compiler writers and processor architects to evaluate the performance of the computer systems. CPU2006 is the improved benchmark version of CPU2000 (Henning, 2000). CPU2006 covers more emerging applications to increase the variety and representation within the suite. Most of these applications have larger resource requirements and are more complex as when compared to the applications supported in CPU2000. Joshi et al., (2006) studied the evolution of CPU2000 and postulates a representative subset of programs from the benchmark suite. Kejariwal et al. (2008) presented an analysis of the behaviour and performance of different programs on CPU2006 and compares them to the programs of CPU2000 using a state-of-the-art production compiler and architecture.

CPU2006 focuses on compute intensive performance. This includes the benchmark on CPU, memory architecture and compiler. The CINT2006 suite measures compute-intensive

integer performance while the CFP2006 suite measures compute-intensive floating point performance. Li et al. (2009b) conducted the performance evaluation and analysis of CPU2006 benchmark suite. The analysis is done on hardware configuration and optimization technologies.

PCMark05 (2010) is another benchmark suite created by FutureMark. PCMark05 supports both system level and component level benchmarking. System benchmarking evaluates the overall performance of the computer system. Component benchmarking, by contrast, measures the performance of individual components such as CPU, memory, graphics and etc. Like other benchmark tools, PCMark05 comprises of separate test suites for CPU, Memory, Graphics and hard disk. In addition, PCMark05 includes a system test suite that generates an overall system benchmark score.

In summary, resource benchmark is another important process to improve the scheduling process. The combination of application benchmark and resource benchmark is able to provide more accurate job and resource information to the scheduler and directly improve the scheduling decision. PerformanceTest 7.0 is selected as the benchmark tool to evaluate the resource performance. This is because PerformanceTest 7.0 can benchmark a resource using a variety of different speed tests. Besides, PerformanceTest 7.0 consists of built-in baseline results that enable the comparison with other resources.

### 4.2.2 Implementation of Resource Benchmarking

The PerformanceTest 7.0 benchmark suite is used as a micro-benchmark to profile resources capabilities such as CPU and memory. Table 4.4 shows the CPU marks obtained

for some of the single CPU and multiple CPUs computer system. The higher the CPU mark obtained, the better is the computer system performance.

**Table 4.4 CPU mark for single CPU and multiple CPUs (PassMark, 2010)**

| CPU Name | CPU Mark |
|---|---|
| [Quad CPU] AMD Opteron 6168 | 23784 |
| [8-Way] Six-Core AMD Opteron 8435 | 22745 |
| [Quad CPU] Intel Xeon X7460 @ 2.66GHz | 18304 |
| [Dual CPU] Intel Xeon X5680 @ 3.33GHz | 17377 |
| [Quad CPU] Intel Xeon X7350 @ 2.93GHz | 16715 |
| [Dual CPU] Intel Xeon X5670 @ 2.93GHz | 15320 |
| [Dual CPU] AMD Opteron 6174 | 15017 |
| [Dual CPU] Intel Xeon X5660 @ 2.80GHz | 14486 |
| [Dual CPU] Intel Xeon W5580 @ 3.20GHz | 13009 |
| [Dual CPU] Intel Xeon X5650 @ 2.67GHz | 13007 |
| [Dual CPU] Intel Xeon W5590 @ 3.33GHz | 12740 |
| [Quad CPU] Quad-Core AMD Opteron 8386 SE | 12335 |
| [Dual CPU] Intel Xeon X5677 @ 3.47GHz | 12117 |
| [Quad CPU] Quad-Core AMD Opteron 8360 SE | 11912 |
| [Dual CPU] Intel Xeon X5560 @ 2.80GHz | 11658 |
| Intel Xeon E5630 @ 2.53GHz | 5006 |
| Intel Xeon X5482 @ 3.20GHz | 5006 |
| Intel Core2 Extreme X9770 @ 3.20GHz | 4998 |
| Intel Xeon X3440 @ 2.53GHz | 4990 |
| [Dual CPU] Intel Xeon X5260 @ 3.33GHz | 4979 |
| AMD Phenom II X6 1035T | 4964 |
| Intel Xeon W3530 @ 2.80GHz | 4964 |
| [Dual CPU] Quad-Core AMD Opteron 2354 | 4930 |
| Intel Xeon W3520 @ 2.67GHz | 4889 |
| Intel Xeon X5460 @ 3.16GHz | 4877 |
| [Dual CPU] Quad-Core AMD Opteron 2350 | 4841 |
| Intel Core i7 X 940 @ 2.13GHz | 4774 |

Table 4.5 shows the average execution time by running the same Web Services with two different parameter values on different computer systems with the CPU marks collected using PerformanceTest 7.0.

**Table 4.5 Benchmark results: average execution time of service 1 running on different resources**

| | Average Execution Time | |
|---|---|---|
| **CPU Mark** | **Parameter = 50000000** | **Parameter = 100000000** |
| 315 | 460.97 | 895.42 |
| 410 | 415.77 | 835.77 |
| 492 | 290.20 | 583.64 |
| 524 | 267.07 | 534.71 |
| 1451 | 181.13 | 360.60 |
| 4733 | 74.37 | 153.57 |

Figure 4.4 depicts that the relationship between the CPU mark and the execution time. From the figure, the average execution time is inversely proportional to the CPU mark. A trend line is added to the graph and a mathematical equation is formed. From the equation, the estimated execution time for the same Web Services running on different resources is computed. The estimated execution time gets more accurate when more historical records are available.

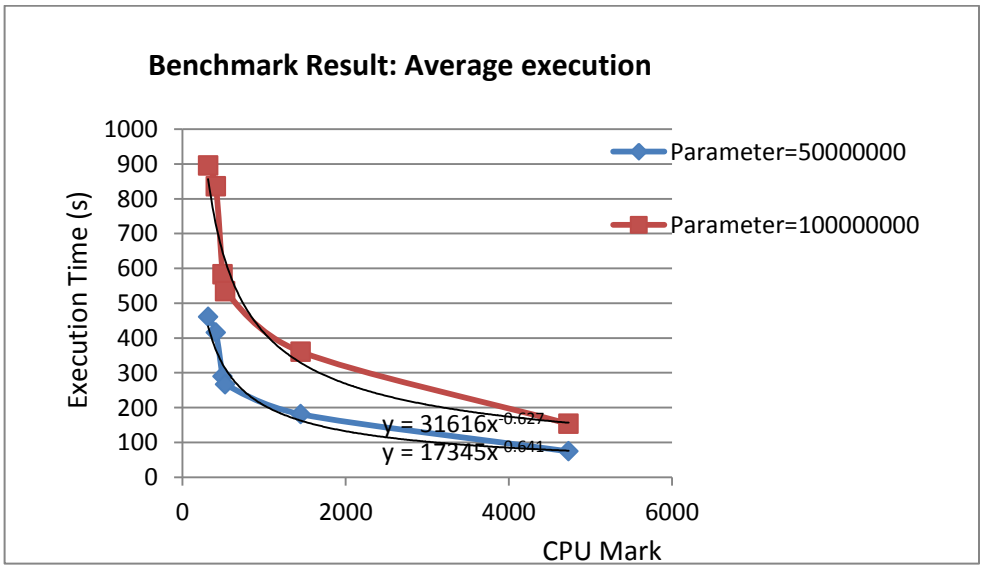


**Figure 4.4 Benchmark result: average execution time of service 1 running on different resources**

The job length is estimated using a combination of application benchmark and resource benchmark. Several test cases which include 1) Random job arrival with random job length; 2) Random job arrival with big gap job length; 3) Random job arrival with job length in Poisson distribution are used to compare the effect of running HSA with and without benchmarking. The same set of experiment settings as in Table 3.2 is used.

Figure 4.5 shows the makespan (in seconds) for HSA with and without benchmarking running on different test cases. Table 4.6 shows the results of the makespan in (second) and percentage of improvement of HSA with benchmarking as compared to HSA with grouping for different test cases

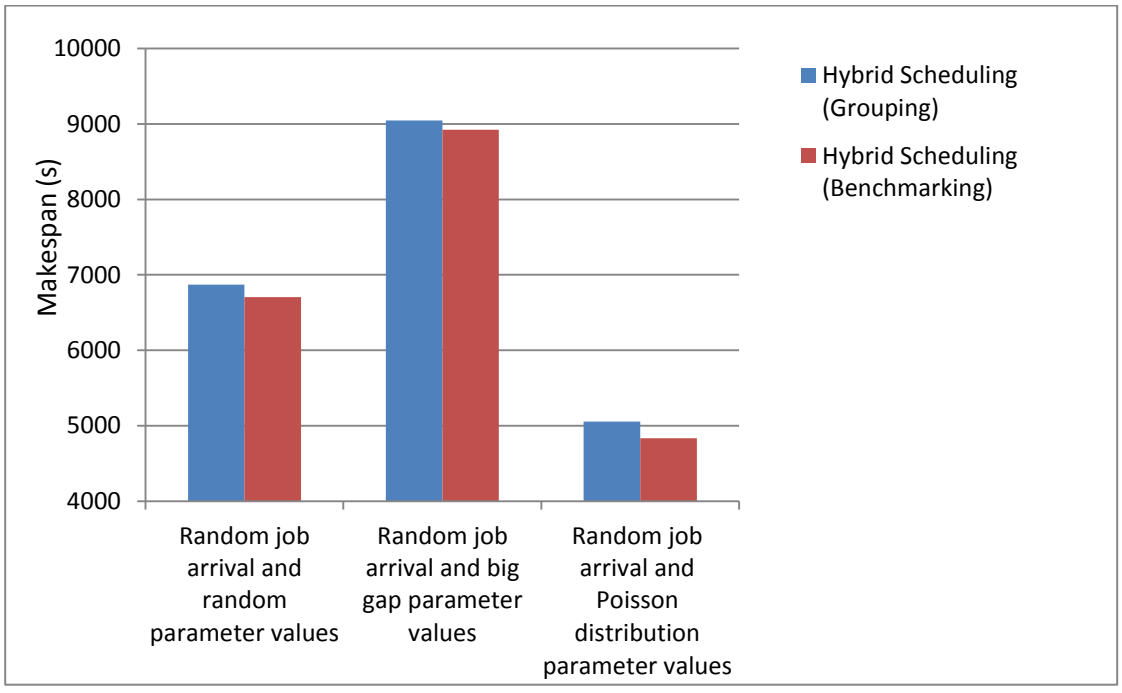


**Figure 4.5 Makespan (second) of HSA with and without benchmarking for different experimental cases**

**Table 4.6 Makespan and percentage of improvement of HSA with benchmarking as compared to HSA with grouping for different test cases**

| Test | Job Description | HSA (Grouping ) | HSA (Benchmarking ) | % improvement of HSA (Benchmarking) as compared to HSA (Grouping) |
|---|---|---|---|---|
| 1 | Random job arrival and random parameter values | 6871.97 | 6706.72 | 2.4 |
| 2 | Random job arrival and big gap parameter values | 9045.27 | 8925.34 | 1.3 |
| 3 | Random job arrival and Poisson distribution parameter values | 5055.39 | 4832.76 | 4.4 |

It is observed from Table 4.6 that HSA with benchmarking performs better than HSA without benchmarking. The improvement of 1% - 4% in terms of makespan indicates the efficiency of benchmarking. The use of benchmarking in job length estimation allows the scheduler to perform sorting at each interval without the needs to group the Web Services. In addition, the job length estimation provides more accurate tasks information to the scheduler. In turn, the scheduler is able to provide better scheduling decision using the application and resource benchmarking as compared to Web Services grouping.

### 4.3 Adaptive Scheduling Algorithm (ASA)

One of the drawbacks of HSA is that the z value is predefined for each experimental testing. Many testing are required in order to get the optimal z value to minimize the makespan.

Besides, the z value is fixed throughout the experiment testing which mean that the same z value is used at each different interval.

In order to resolve the above issue, an Adaptive Scheduling Algorithm (ASA) is proposed. ASA enhances HSA by using a combination of resource benchmark and application benchmark to estimate the job length and uses a dynamic threshold value, z, at each interval to make scheduling decision to dispatch task either to primary or secondary resources. At each interval, the mean value is computed and the heuristic method is used to get the best z value to achieve the optimal makespan.

Figure 4.6 depicts the main components of ASA. The benchmarker is added to provide the application-specific benchmark and micro-benchmark or resource benchmark. The micro-benchmark is used to profile resources capabilities while the application-specific benchmark measures the performance of resources when executing different types of jobs. The combination of micro-benchmark and application benchmark provides the estimation of job length.

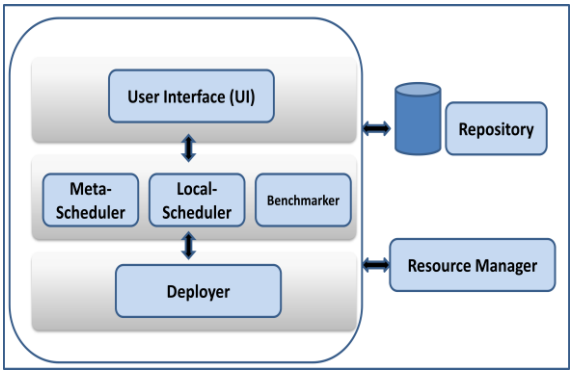


**Figure 4.6 Main components of ASA**

In chapter 3, HSA has to group the Web Services at each interval before making the scheduling decision. ASA enhances the HSA by accepting any Web Services request at each interval. Thus, the job request has been redefined as $I_i$, where $I_i$ consists of a set of different task at interval i. Each task can belong to different Web Services.

$$I_i = \{T_{i,1}, T_{i,2}, \ldots, T_{i,n}\}, 0 \leq i \leq n \text{ and } i, n \in \mathbb{N} \quad (4.1)$$

The estimated execution time for each task at time interval i is computed by using the benchmark results collected from the application specific benchmark and micro-benchmark. The estimated execution time for task $T_{i,j}$ is given as $e_{i,j}$. A sorted list, $K_i$, is then generated by sorting the estimated execution time for each task at interval i.

$$K_i = \{e_{i,j}, e_{i,(j+1)}, \ldots, e_{i,n}\}, \text{ where } e_{i,j} > e_{i,n} \text{ and } j<n \quad (4.2)$$

The process continues with the generation of sorted primary resource, $P_i$ for each $T_i$, and sorted secondary resource, $Q_i$ for each $T_i$. The process is followed by the computation of the mean value.

$$mean_i = \sum_1^n e_{i,j} / n, \text{ where } mean_i \text{ is the mean value for } K_i \quad (4.3)$$

The mean value, $mean_i$, is used to determine whether the task, $T_{i,j}$, is assign to the resources in $P_i$ or $Q_i$ based on the pseudo code below:

```
if  l_{i,j}  > mean_i * z then
  allocate T_{i,j} to P_i that returns the minimum completion time
else
  allocate T_{i,j} to Q_i that returns the minimum completion time
end if
```

ASA enhances HSA by introducing the dynamic value z for each interval. The best z value is used to generate a schedule list with shortest makespan. Figure 4.7 depicts the proposed scheduling algorithm.

Input:
    $I_i$, a set of different task request at interval i
    $p_m$ or $q_n \in S_i$, , where $S_i$ is a set of resources in site i
Output
    A schedule of $T_{i,j}$ into $p_m$ or $q_n$

// estimate the task length at each interval
∀ task request, $I_i$, in waiting queue
  estimate the execution time, $e_{i,j}$
  Sort the task $T_{i,j}$ in descending order based on the estimated execution time, $e_{i,j}$
  Generate a sorted task requests list $K_i$
  Compute $mean_i$ from $K_i$

  ∀ servers and clusters in site I  // generate the primary resource list
    Select potential resources and generate sorted primary resource list, $P_i$

  ∀ desktop computers in site I  I  // generate the secondary resource list
    Select available desktop computers and generate sorted secondary resource list, $Q_i$

  // compare the multiplication of z and mean with the parameter value to
  // determine whether to schedule the task to primary or secondary resource
  ∀ Task, $T_{i,j}$
    if $l_{i,j} > mean_i * z$
      allocate $T_{i,j}$ to $p_m$ that returns the minimum completion time
    Otherwise
      allocate $T_{i,j}$ to $q_n$ that returns the minimum completion time
    Repeat the experiment for each interval,
    With the dynamic value, z $\pm 0.1$ to get the minimum makespan

**Figure 4.7 Proposed ASA**

### 4.4 Experiment Setup

An experimental testbed with the implementation of the same components as chapter 3 is developed to execute the testing of services in Grid and Cloud environment. An extra

component, the Benchmarker which is used to estimate job length for each task is added to the testbed. The same set of Web Services and resources is used. The test cases used to verify the correctness of the proposed ASA, MIN-MIN with benchmarking and MAX-MIN implementations are presented in Appendix A.2.

To evaluate the performance of the proposed algorithm, the same set of resources and job parameters are used and is shown in Table 4.7. A set of evaluations with similar settings as Table 4.7 is carried out to compare the performance of the proposed ASA against MIN-MIN and MAX-MIN scheduling algorithm. The performance metric used for the comparison is makespan.

**Table 4.7 A predefined set of resources and job parameters**

| Description | Value Range |
|---|---|
| Number of sites | 3 – 5 |
| Number of hosts per site | 20 – 50 |
| Number of CPU | 1 – 4 |
| Number of core per CPU | 1 – 8 |
| CPU speed | 0.8 – 3.2 GHz |
| Number of tasks | 300 – 500 |
| Execution time of task (second) | 1 – 1000 |
| Type of Web Services | 1 – 10 |

Several test cases which include 1) Random job arrival with random job length; 2) Random job arrival with big gap job length; 3) Random job arrival with job length in Poisson distribution; 4) Poisson arrival with random job length; 5) Poisson arrival with big gap job length; 6) Poisson arrival with job length in Poisson distribution are used for the experiment.

## 4.5 Result and Analysis

Table 4.8 – 4.13 show the results of the makespan (in seconds) for different experimental cases using different scheduling algorithms. Figure 4.8 shows the results from Table 4.8 – 4.13 in graphical form.

**Table 4.8 Makespan (seconds) of different algorithms for random job arrival and random job length**

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Random job arrival and random job length (Benchmarking) - MIN-MIN | 7455.17 | 7565.56 | 7572.55 | 7505.72 | 7503.62 | 7520.52 |
| 2 | Random job arrival and random job length (Benchmarking) - MAX-MIN | 6801.52 | 6841.72 | 6823.08 | 6823.17 | 6829.77 | 6823.85 |
| 3 | Random job arrival and random job length (Benchmarking) - HSA (z=1.1) | 6898.87 | 6875.72 | 6945.32 | 6892.96 | 6881.42 | 6898.86 |
| 4 | Random job arrival and random job length (Benchmarking) - HSA (z=1) | 6734.28 | 6691.77 | 6727.13 | 6708.31 | 6672.12 | 6706.72 |
| 5 | Random job arrival and random job length (Benchmarking) - HSA (z=0.9) | 6923.35 | 6889.43 | 6819.14 | 6914.30 | 6876.18 | 6884.48 |
| 6 | Random job arrival and random job length (Benchmarking) - ASA | 6415.67 | 6405.90 | 6462.27 | 6494.87 | 6460.11 | **6447.76** |

**Table 4.9 Makespan (seconds) of different algorithms for random job arrival and big gap job length**

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Random job arrival and big gap job length (Benchmarking) - MIN-MIN | 10523.12 | 10542.85 | 10610.48 | 10465.15 | 10413.07 | 10510.93 |
| 2 | Random job arrival and big gap job length (Benchmarking) - MAX-MIN | 9102.12 | 9188.43 | 9082.90 | 9124.49 | 9153.91 | 9130.37 |
| 3 | Random job arrival and big gap job length (Benchmarking) - HSA (z=1.5) | 10071.43 | 9999.51 | 9975.81 | 10118.77 | 9956.48 | 10024.40 |
| 4 | Random job arrival and big gap job length (Benchmarking) - HSA (z=1.4) | 8791.15 | 9033.57 | 8784.83 | 9040.88 | 8976.26 | 8925.34 |
| 5 | Random job arrival and big gap job length (Benchmarking) - HSA (z=1.3) | 9292.49 | 9228.70 | 9144.49 | 9229.45 | 9123.67 | 9203.76 |
| 6 | Random job arrival and big gap job length (Benchmarking) - ASA | 8707.17 | 8626.44 | 8606.46 | 8682.07 | 8612.51 | **8646.93** |

**Table 4.10 Makespan (seconds) of different algorithms for random job arrival and job length in Poisson distribution**

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Random job arrival and job length in Poisson Distribution (Benchmarking) - MIN-MIN | 5602.55 | 5615.41 | 5572.31 | 5578.81 | 5592.01 | 5592.22 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | Random job arrival and job length in Poisson Distribution (Benchmarking) - MAX-MIN | 4988.60 | 4976.24 | 4946.25 | 4985.78 | 4952.73 | 4969.92 |
| 3 | Random job arrival and job length in Poisson Distribution (Benchmarking) - HSA (z=1) | 4948.82 | 4991.30 | 4972.15 | 4941.71 | 4970.90 | 4964.98 |
| 4 | Random job arrival and job length in Poisson Distribution (Benchmarking) - HSA (z=0.9) | 4793.36 | 4820.59 | 4850.89 | 4817.96 | 4881.01 | 4832.76 |
| 5 | Random job arrival and and job length in Poisson Distribution (Benchmarking) - HSA (z=1) | 4906.10 | 4972.96 | 4923.15 | 4989.31 | 4945.24 | 4947.35 |
| 6 | Random job arrival and job length in Poisson Distribution (Benchmarking) - ASA | 4691.94 | 4697.17 | 4708.84 | 4651.38 | 4753.50 | **4700.57** |

## Table 4.11 Makespan (seconds) of different algorithms for Poisson arrival ( $\lambda$ =50) and random job length

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Poisson job arrival (λ=50) and random job length (Benchmarking) - MIN-MIN | 6919.68 | 6816.69 | 7024.63 | 6947.40 | 6856.05 | 6912.89 |
| 2 | Poisson job arrival (λ=50) and random job length (Benchmarking) - MAX-MIN | 6070.11 | 6058.11 | 6048.28 | 6007.66 | 6036.97 | 6044.23 |
| 3 | Poisson job arrival (λ=50) and random job length (Benchmarking) - HSA (z=1.2) | 6527.63 | 6573.01 | 6548.98 | 6481.88 | 6489.22 | 6524.14 |
| 4 | Poisson job arrival (λ=50) and random job length (Benchmarking) - HSA (z=1.1) | 5943.49 | 5893.41 | 5834.64 | 5903.22 | 5892.53 | 5893.46 |
| 5 | Poisson job arrival (λ=50) and random job length (Benchmarking) - HSA (z=1) | 6157.79 | 6158.02 | 6064.13 | 6086.96 | 6176.36 | 6128.65 |
| 6 | Poisson job arrival (λ=50) and random job length (Benchmarking) - ASA | 5711.10 | 5733.70 | 5683.70 | 5788.55 | 5697.66 | **5722.94** |

## Table 4.12 Makespan (seconds) of different algorithms for Poisson arrival ( $\lambda$ =50) and big gap job length

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Poisson job arrival (λ=50) and big gap job length (Benchmarking) - MIN-MIN | 11331.86 | 11115.91 | 11098.14 | 11353.72 | 11224.34 | 11224.79 |
| 2 | Poisson job arrival (λ=50) and big gap job length (Benchmarking) - MAX-MIN | 10234.83 | 10964.71 | 10038.74 | 9995.73 | 10942.95 | 10435.39 |
| 3 | Poisson job arrival (λ=50) and big gap job length (Benchmarking) - HSA (z=1.4) | 10544.57 | 10703.82 | 10567.45 | 10417.34 | 10641.86 | 10575.01 |
| 4 | Poisson job arrival (λ=50) and big gap job length (Benchmarking) - HSA (z=1.3) | 9924.75 | 9988.45 | 9986.87 | 9980.12 | 9995.02 | 9975.04 |
| 5 | Poisson job arrival (λ=50) and big gap job length (Benchmarking) - HSA (z=1.2) | 10414.41 | 10378.90 | 10299.76 | 10035.94 | 10415.50 | 10308.90 |
| 6 | Poisson job arrival (λ=50) and big gap job length (Benchmarking) - ASA | 9833.84 | 9858.60 | 9872.14 | 9880.36 | 9809.01 | **9850.79** |

**Table 4.13 Makespan (seconds) of different algorithms for Poisson arrival ( λ =50) and job length in Poisson distribution (λ1=40000000, λ2=4000, λ3=50000)**

| Test | Job Description | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| 1 | Poisson job arrival (λ=50) and job length in Poisson Distribution (λ1=40000000, λ2=4000, λ3=50000) (Benchmarking) - MIN-MIN | 5473.76 | 5517.44 | 5450.79 | 5554.15 | 5470.79 | 5493.39 |
| 2 | Poisson job arrival (λ=50) and job length in Poisson Distribution (λ1=40000000, λ2=4000, λ3=50000) (Benchmarking) - MAX-MIN | 4836.20 | 4826.70 | 4858.98 | 4889.84 | 4889.32 | 4860.21 |
| 3 | Poisson job arrival (λ=50) and job length in Poisson Distribution (λ1=40000000, λ2=4000, λ3=50000) (Benchmarking) - HSA (z=1) | 5003.34 | 5016.55 | 5010.42 | 5009.08 | 4965.55 | 5000.99 |
| 4 | Poisson job arrival (λ=50) and job length in Poisson Distribution (λ1=40000000, λ2=4000, λ3=50000) (Benchmarking) - HSA (z=1.1) | 4755.77 | 4749.37 | 4726.71 | 4761.11 | 4772.14 | 4753.02 |
| 5 | Poisson job arrival (λ=50) and and job length in Poisson Distribution (λ1=40000000, λ2=4000, λ3=50000) (Benchmarking) - HSA (z=0.9) | 4863.31 | 4799.67 | 4832.08 | 4829.31 | 4851.44 | 4835.16 |
| 6 | Poisson job arrival (λ=50) and job length in Poisson Distribution (λ1=40000000, λ2=4000, λ3=50000) (Benchmarking) - ASA | 4516.07 | 4609.82 | 4554.55 | 4598.30 | 4567.62 | **4569.27** |

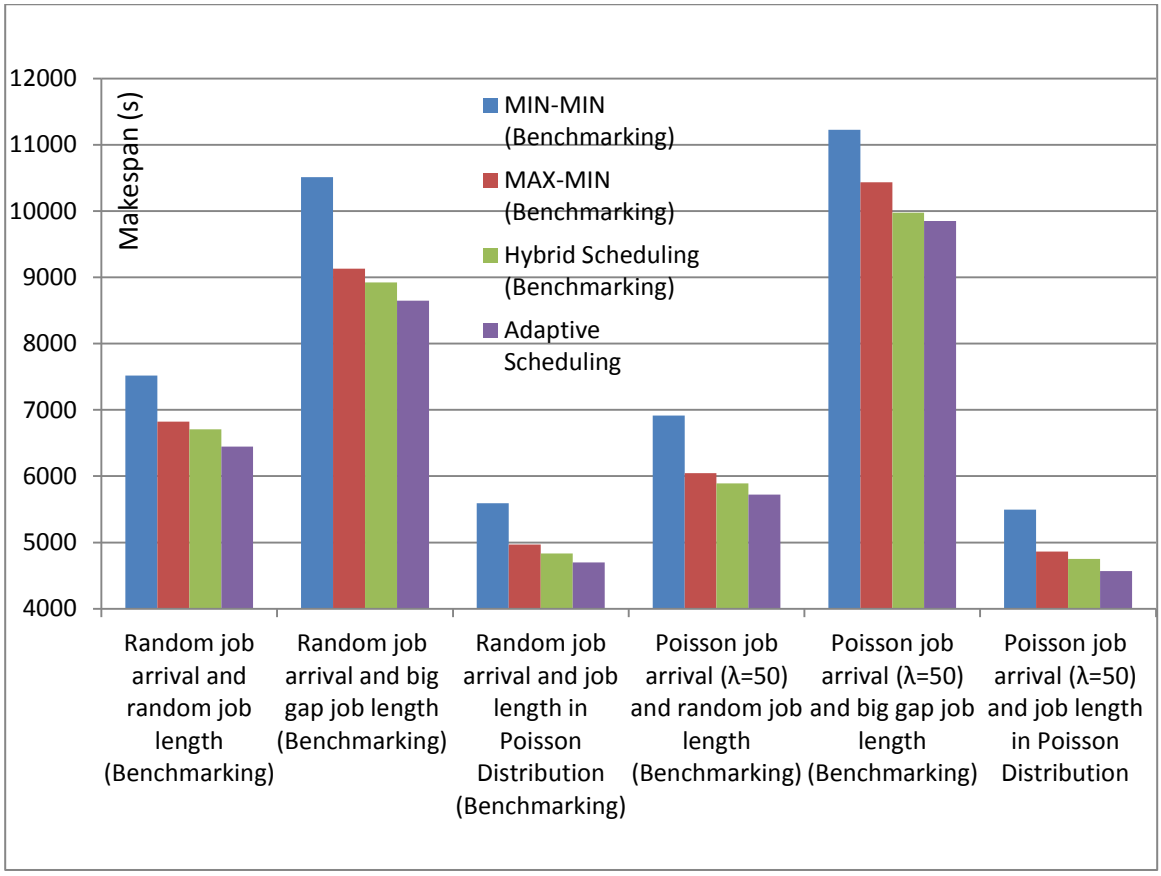


**Figure 4.8 Makespan (seconds) of different algorithms with benchmarking for different experimental cases**

Table 4.8 - Table 4.10 shows that ASA yields the best results compared to the other scheduling algorithms, for different types of job length distributions and random arrival time. It is also observed from Table 4.11 – 4.13 that ASA outperforms MIN-MIN with benchmarking, MAX-MIN with benchmarking and HSA with benchmarking by a significant margin for different types of job length distributions when the jobs arrival is Poisson.

Table 4.14 shows the results of the makespan (in seconds) and the percentage of improvement. It is observed that ASA outperforms MIN-MIN with benchmarking by between 12% - 17% in terms of makespan. Table 4.14 also shows that the improvement of ASA as compared to MAX-MIN with benchmarking is by between 5% - 6% in terms of makespan. The superior performance of ASA is attributed to the ability to determine the mean value to use at each interval and to dynamically generate the z value that controls the decision to schedule tasks to primary resources or secondary resources.

**Table 4.14 Makespan (seconds) and percentage of improvement of ASA as compared to three different scheduling algorithms with benchmarking for different test cases**

| Test | Job Description | MIN-MIN (Benchmarking) | MAX-MIN (Benchmarking) | HSA (Benchmarking) | ASA | % improvement of ASA as compared to MIN-MIN (Benchmarking) | % improvement of ASA as compared to MAX-MIN (Benchmarking) | % improvement of ASA as compared to HSA (Benchmarking) |
|---|---|---|---|---|---|---|---|---|
| 1 | Random job arrival and random job length (Benchmarking) | 7520.52 | 6823.85 | 6706.72 | 6447.76 | 14.3 | 5.5 | 3.9 |
| 2 | Random job arrival and big gap job length (Benchmarking) | 10510.93 | 9130.37 | 8925.34 | 8646.93 | 17.7 | 5.3 | 3.1 |
| 3 | Random job arrival and job length in Poisson | 5592.22 | 4969.92 | 4832.76 | 4700.57 | 15.9 | 5.4 | 2.7 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Distribution (Benchmarking) | | | | | | | |
| 4 | Poisson job arrival (λ=50) and random job length (Benchmarking) | 6912.89 | 6044.23 | 5893.46 | 5722.94 | 17.2 | 5.3 | 2.9 |
| 5 | Poisson job arrival (λ=50) and big gap job length (Benchmarking) | 11224.79 | 10435.39 | 9975.04 | 9850.79 | 12.2 | 5.6 | 1.2 |
| 6 | Poisson job arrival (λ=50) and job length in Poisson Distribution | 5493.39 | 4860.21 | 4753.02 | 4569.27 | 16.8 | 6.0 | 3.9 |

From Table 4.14, ASA only shows a slight improvement of 1% - 4% over HSA with benchmarking. However, the dynamic z value provides the ability to adapt to different situations. A fixed z value is no longer required to be predefined for each experiment. Using a z value that changes dynamically at each interval makes the adaptive mechanism more feasible and practical for a heterogeneous environment and delivers better schedules as compared to HSA.

## 4.6 Chapter Summary

This chapter presents Adaptive Scheduling Algorithm (ASA) which enhances the HSA through benchmarking as well as the adaptive mechanism. In ASA, a two level benchmarking is used to benchmark the job requests at each interval. The algorithm uses the application specific benchmark and resource benchmark to estimate the job length. Then, an adaptive mechanism is applied to the scheduling decision at each interval where a dynamic value z, is used to identify whether the job is assigned to primary or secondary resources.

ASA is then compared to the scheduling algorithms such as MIN-MIN with benchmarking, MAX-MIN with benchmarking and HSA with benchmarking. The experimental results show that ASA minimizes the makespan by 5% - 17%. In addition, ASA also reduces the makespan by 1 - 4% when compared to HSA with benchmarking. Although this is a small improvement, the dynamic nature of ASA does not require a fix z value for different test cases. Hence, this mechanism is more feasible as compared to HSA. ASA is adaptable to different test cases and provides a better scheduling decision in the Grid and Cloud environment. In summary, ASA is able to maximize resource utilization and minimize makespan.

Besides the job scheduling problems discuss above, Quality of Service (QoS) is becoming a big concern in the Grid and Cloud environment. Thus, in the next chapter, the QoS issues in the Grid and Cloud environment are discussed. Also, an enhanced ASA is proposed to maximize reliability and profit while guaranteeing the users QoS requirements.

# CHAPTER 5 ADAPTIVE QoS SCHEDULING ALGORITHM (AQoSSA)

This chapter presents the solution that guarantees end users QoS requirements and provides cost management for service providers in Grid and Cloud environment. The first section details QoS in Grid and Cloud Computing. In the second section, the Service Level Agreement (SLA) is presented. The third section presents the AQoSSA and the rescheduling mechanism. The final section describes the experiment setup and evaluates the performance of AQoSSA.

## 5.1 QoS in Grid and Cloud Computing

Grid and Cloud Computing have emerged as the new computing paradigms that enable resources sharing and dynamic allocation of resources for various applications. These technologies involved various resources distributed in multiple domains. As these technologies provide services to many users from distributed locations, the ability to guarantee QoS is becoming critical. Besides the growth in the number of users, different users having different QoS requirements make the guaranteeing of QoS in Grid and Cloud extremely challenging.

Service providers are not only providing the functional requirements of the services to the end users, they need to satisfy the non-functional requirements of the services. To be successful in Grid and Cloud, the ability to deliver guaranteed QoS services is crucial. In order to guarantee QoS in Grid and Cloud Computing, various important QoS requirements

such as availability, scalability, deadline, cost and security are identified and discussed below.

### 5.1.1 Availability

In a dynamic Grid and Cloud environment, the most important criterion for QoS is the availability of service. Availability is the probability that a service is available at a particular moment. Availability can be formulated as

$$A_i = t_i / T \quad (5.1)$$

where, $t_i$ is the total uptime of service i and T is the total time of evaluation. If the value is one, the service is always available. The availability of a service can be affected by resources going offline due to hardware failure, software update or rebooting.

### 5.1.2 Scalability

Since service providers serve many users, the service providers must have the ease to scale upwards or downwards the resource infrastructure to facilitate the varying number of users and users' requirements. This is to ensure the QoS requirements are met as well as to provide better resource optimization. Based on the Horizontal Cloud and Vertical Cloud approaches proposed by (Mei et al., 2008), the resources can be scaled and provided dynamically on demand. In Cloud computing, virtualization technology is one of the important technologies that provides scalability and optimal use of resources. With scalability, end users have higher chances of accessing services deployed in Grid and Cloud environment at any time and from anywhere.

However, scalability is turning out to be a very challenging issue for service providers. If the Grid or Cloud over provide resources, the over-provisioning leads to under-utilization during periods of low demand. On the other hand, in the event of under-provisioning, some of the end users' requests will be rejected. Furthermore, Cloud computing has to be highly elastic in scalability where resources can be scaled up or down within a given time.

### 5.1.3 Deadline

The deadline or response time is a particular point in time which is usually provided by an end user to specify the required job completion time. Being measurable, this parameter is usually explicitly specified in the SLA. The deadline parameter is usually used in Tail-Distribution based SLA (TD-SLA) since this QoS parameter is measurable in terms of the number of job completions within a given deadline. Usually, for TD-SLA, the terms of the SLA are considered to have been complied when the deadline is within a specified percentage range from the contracted deadline. For example, the deadline is *D* and an over run of 5% is acceptable per the SLA. The service provider will have to pay a penalty if the execution time exceeds *1.05D*.

### 5.1.4 Cost

Grid and Cloud Computing offer the end users the means to utilize the computational resources as well as the applications on a pay-per-use basis. In general, two different cost plans are available, namely, the on-demand plan and the reservation plan. The on-demand plan allows end users to secure the right amount of resources dynamically at any time while the reservation plan requires the end users to reserve the resources within a specific time slot. Usually, the reservation plan is priced cheaper than the on-demand plan. It is easier for

the service providers to cater for sufficient resources under the reservation plan since the required resources are made known at the time of reservation. However, for the on-demand plan, it is difficult for the service providers to estimate the amount of resources required giving rise to under-provisioning or over-provisioning. Thus, the cost of on-demand plan is usually higher than the reservation plan.

Chaisiri et al. (2011) proposed an Optimal Cloud Resource Provisioning (OCRP) algorithm to make provision for resources offered by multiple Cloud service providers. The algorithm is able to provide computing resources in multiple provisioning stages as well as offering different provision plans. Truong and Dustdara (2010) proposed a service cost model for estimating, monitoring and analysing the costs associated with applications in Cloud Computing under different scenarios.

In Grid and Cloud Computing, users only have to pay for the services that are used. This on-demand access significantly reduces the end users infrastructure cost as well as speeds up the job processing. Usually, to reduce cost, some end-users are willing to settle for a lower response time. As such, some end users will select the reservation plan over the on-demand plan.

Most service providers offer different levels of services at different prices to meet the varied demands and needs of the end users. The main objective of most service providers is to maximize their profits through the optimal usage of their resource infrastructures. The profit is computed based on the total money earned from each services running for a given period of time after apportioning the penalty incurred for violation of the SLA. Thus, a

service provider must offer competitive and attractive terms and prices for the services provided while minimizing any penalties for failures to uphold the terms and conditions in the SLAs. This has led to many research studies and mechanisms on cost management.

Hyun Jin et al. (2010) used the profit model that includes revenue and cost functions to compute the total profit gained. Maci et al. (2010) defined different QoS categories, namely, Gold, Silver and Bronze to differentiate the level of services provided. The Gold category is given the highest priority as compared to Silver and Bronze categories. Each category will have a different cost and price associated with it. Yeoa et al. (2010) proposed an autonomic pricing mechanism that dynamically changes the pricing parameters' values to meet the user requirements and maximize profit. Lee et al. (2010) proposed a pricing model using processor-sharing and applying the model to composite services with dependency considerations.

### 5.1.5 Security

Security and privacy are some of the primary concerns in Grid and Cloud Computing. Since Grid and Cloud computing environments are multi-domains, different security, privacy and trust requirements are implemented. Service providers and end users must share the responsibilities for the security and privacy in this environment. The service providers must guarantee that the security features are trustable to the end users. The solution provided by service providers must be efficient and effective against security and privacy risks. It is critical to have the mechanisms to ensure that only authorized entities can gain access to their data. Takabi et al. (2010) explored the lapses in Cloud security and privacy and suggested the solutions to create a trustworthy Cloud Computing environment.

Once the QoS requirements are identified, the requirements must be documented and agreed upon by the service providers and the end users in the form of an SLA. The next section will discuss SLA which defines the scope of the services provided and provisioning of resources.

### 5.2 Service Level Agreement

In Grid and Cloud Computing, the interaction between end users and service providers is negotiated and contracted through the Service Level Agreement (SLA). SLA is a document that includes a description of the agreed services, service level objectives, guarantees and actions and remedies for all cases of violations (Andrieux et al., 2005). SLA is a formal contract between the service provider and the end user that specifies the required and acceptable QoS metric levels as well as penalties in the event of service violations. An effective SLA will ensure that the service providers and users interests are protected within the agreed terms and conditions framework.

In general the SLA can be divided into Hard SLA and Soft SLA (Hyun Jin et al., 2010). In a Hard SLA, the contract is violated as soon as the QoS requirement is not met while a Soft SLA has a varied level of violation that corresponds to agreed levels of service. The soft SLA specifies the percentage of SLA violations that can be tolerated within a predefined time interval before the service provider is legally culpable to pay for failing to deliver the prescribed services. Hence, a good SLA enactment strategy is necessary to avoid costly penalties for violating the terms and conditions spelt out on the SLA.

Service providers use SLAs to define trust, the level of QoS provided as well as service charges. A service provider can optimize the resource usages based on the agreed terms and conditions in a SLA. In order to comply with the SLA, a service provider must be capable of monitoring the infrastructure resources. In addition, a service provider must be able to predict the service performances as well as the number of end users to assure the QoS while avoiding over-provisioning. However, due to the heterogeneous services and resources, a dynamic provisioning of resources to meet SLA is very challenging.

End users make use of the SLA to obtain the needed level of QoS and to maintain the acceptable provisioning of the required services. The end users pay only for the use of resources and services within the SLA framework. Usually, the end users are more concerned about the response time and the service output.

The SLA is usually formulated through a process of negotiation between the service providers and the end users. Currently, the most common negotiation method is through direct interaction between end users and service providers. The service providers will define all the SLA criteria with pricings while the end users will review all the SLA terms and conditions. If agreeable, the SLA is accepted and contracted. Otherwise, renegotiation or termination is initiated.

In the current marketplace, service providers offer multiple services to many end users at different locations making SLAs in Grid and Cloud Computing all the more important. There are currently many research studies that are looking into SLAs in relation to Grid and Cloud Computing. Ferretti et al. (2010) proposed the design of a middleware architecture

that enables SLA-driven dynamic configuration, management and optimization of Cloud resources and services to meet the QoS requirements. Alhamad et al. (2010) proposed a conceptual SLA framework for Cloud Computing environment. Different SLA metrics are defined for different service layers. Fito et al. (2010) proposed a Cloud Hosting Provider that uses outsourcing technique to provide scalability, high availability and maximizing the revenue earned by service provider through the analysis of SLA and the employment of an economic model.

Reig et al. (2010) proposed a prediction system to determine the minimum amount of resources required to be committed to meet a job deadline. The authors used an analytical predictor and self-adjusting predictor to predict job resource requirements which the CPU share and the amount of memory required to execute a job within its deadline. This helps the service providers to fully utilize the resources while avoiding any SLA violations. Boloor et al. (2010) proposed a heuristics based request scheduling to minimize the overall penalty charged for SLA violations. Usually, the SLA has some additional time allowances above the deadline before a penalty is incurred.

In summary, the ability to deliver guaranteed QoS service in Grid and Cloud environment is crucial. Since the service providers offer services on a pay-per-use model, a profit model that is based on resource usage is more appropriate as compared to rental model where resource is charge based on daily or monthly term.

## 5.3 Adaptive Quality of Service Scheduling Algorithm (AQoSSA)

After reviewing various QoS requirements as well as SLAs, the AQoSSA is proposed. A profit model is proposed in AQoSSA to measure the profit generated by the service providers. Subsequently, AQoSSA will schedule the jobs base on the QoS requirements specified in SLA. The objective function of AQoSSA is to maximize the reliability and profitability of the service providers while guaranteeing the user QoS requirements.

### 5.3.1 Profit Model

In arriving at AQoSSA, the revenue generated by running the services on different resources is first identified. Amazon EC2 and Windows Azure offer different pricing schemes for the end users. In this research, the unit of revenue is measured by using the CPU benchmark of the resources. Table 5.1 shows the average execution time for different resources running on different parameter values. The parameter values used are based on the same Web Services that has been benchmarked in Chapter 4. The parameter value and the average execution time are linearly proportional to one another when running on the same resources.

**Table 5.1 Average Execution Time (second)**

| RID | CPU Benchmark | Parameter Values | | | | |
|---|---|---|---|---|---|---|
| | | 5000000 | 10000000 | 50000000 | 100000000 | 500000000 |
| 1 | 315 | 45.43 | 90.03 | 460.97 | 892.33 | 4471.10 |
| 2 | 410 | 42.30 | 83.57 | 415.77 | 834.13 | 4154.20 |
| 3 | 492 | 30.47 | 59.17 | 290.20 | 581.97 | 2895.47 |
| 4 | 524 | 26.67 | 53.00 | 267.07 | 534.27 | 2690.20 |
| 5 | 1451 | 18.23 | 36.47 | 181.13 | 360.50 | 1807.07 |
| 6 | 4089 | 9.83 | 22.53 | 94.53 | 188.13 | 925.83 |
| 7 | 4733 | 9.37 | 19.27 | 74.37 | 152.93 | 727.77 |

Using the CPU benchmark as a measure of revenue, the revenue is defined as

$$revenue = (CPUBenchmark / 10000) * ExecutionTime \quad (5.2)$$

Table 5.2 shows the revenue for different resources running on different input. The Resource ID 6 generates the highest revenue for the largest input. This fact shows that the highest CPU benchmark does not necessary will generate the highest revenue.

**Table 5.2 Revenue (unit)**

| RID | CPU Benchmark | Revenue | | | | |
|---|---|---|---|---|---|---|
| | | 5000000 | 10000000 | 50000000 | 100000000 | 500000000 |
| 1 | 315 | 1.43 | 2.84 | 14.52 | 28.11 | 140.84 |
| 2 | 410 | 1.73 | 3.43 | 17.05 | 34.20 | 170.32 |
| 3 | 492 | 1.50 | 2.91 | 14.28 | 28.63 | 142.46 |
| 4 | 524 | 1.40 | 2.78 | 13.99 | 28.00 | 140.97 |
| 5 | 1451 | 2.65 | 5.29 | 26.28 | 52.31 | 262.21 |
| 6 | 4089 | 4.02 | 9.21 | 38.65 | 76.93 | 378.57 |
| 7 | 4733 | 4.43 | 9.12 | 35.20 | 72.38 | 344.45 |

Hence, the grouping of resources is proposed to derive a relationship between the CPU benchmark and revenue. Table 5.3 shows the mean CPU Benchmark and the relative value of CPU Benchmark from the mean. The mean CPU Benchmark is 1716.3 and the standard deviation of the CPU Benchmark is 1888.17. It is observed from Table 5.3, the resources can be divided into three groups.

**Table 5.3 Mean CPU Benchmark and relative value of CPU Benchmark from mean**

| RID | CPU Benchmark | Mean | \| CPU Benchmark – mean \| |
|---|---|---|---|
| 1 | 315 | 1716.3 | 1401.3 |
| 2 | 410 | 1716.3 | 1306.3 |
| 3 | 492 | 1716.3 | 1224.3 |
| 4 | 524 | 1716.3 | 1192.3 |
| 5 | 1451 | 1716.3 | 265.3 |
| 6 | 4089 | 1716.3 | 2372.7 |
| 7 | 4733 | 1716.3 | 3016.7 |

Figure 5.1 shows the revenue vs CPU Benchmark. It is also observed that from Figure 5.1, the resources can be divided into 3 groups using the k-mean method.

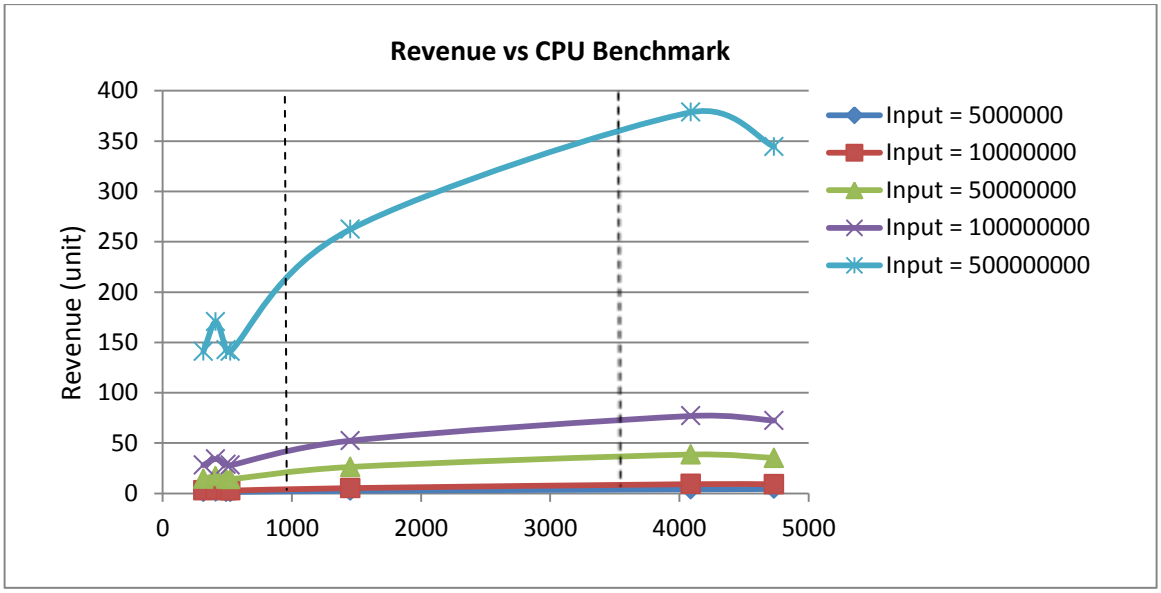


**Figure 5.1: Revenue vs. CPU Benchmark**

Table 5.4 shows the modified revenue derived from the average CPU Benchmark after k-mean grouping. As shown in Table 5.4, after the grouping, group 3 with highest CPU benchmark generates more revenue as compared to the groups with lower CPU benchmark.

**Table 5.4 Modified Revenue (unit)**

| Group | RID | CPU Benchmark | Average CPU Benchmark | Revenue | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 315 | 435.25 | 1.58 | 3.11 | 15.60 | 30.93 | 154.63 |
| | 2 | 410 | | | | | | |
| | 3 | 492 | | | | | | |
| | 4 | 524 | | | | | | |
| 2 | 5 | 1451 | 1451.00 | 2.65 | 5.29 | 26.28 | 52.31 | 262.21 |
| 3 | 6 | 4089 | 4411.00 | 4.23 | 9.22 | 37.25 | 75.22 | 364.70 |
| | 7 | 4733 | | | | | | |

After the resource grouping is completed, the revenue function of the resource is derived. If $RP_i$ is a set of revenue per second for resources in $i^{th}$ site, then

$$RP_i = \{ rp_{i,1}, rp_{i,2}, \ldots, rp_{i,n}\} \; 1 \leq i \leq n \text{ and } i, n \in \mathbb{N} \tag{5.3}$$

$$rp_{i,n} = AVE(CB_{i,n}) / 10000 \tag{5.4}$$

where $AVE(CB_{i,n})$ is the average CPU Benchmark for resource n in $i^{th}$ site that is grouped using the k-mean.

In AQoSSA, each task from equation 4.1 is associated with a Class of Service (CoS), k, and a deadline, $D(T_{i,j,k})$, that is estimated base on the waiting time, execution time and CoS.

$$I_i = \{T_{i,1,k}, T_{i,2,k}, \ldots, T_{i,n,k}\}, 0 \leq i \leq n \text{ and } i, n, k \in \mathbb{N} \tag{5.5}$$

where k represents the classes of services (Gold, Silver and Bronze) which is 1,2 or 3 in this research.

The revenue function, $Rev(T_{i,j,k}, t)$ is given as

$$Rev(T_{i,j,k}, t) = \begin{cases} rp_j * t, & t \leq D(T_{i,j,k}) \\ (2 * rp_j * D(T_{i,j,k})) - (rp_j * t), & D(T_{i,j,k}) < t \leq (1 + \alpha) * D(T_{i,j,k}) \\ -rp_j * D(T_{i,j,k}), & t > (1 + \alpha) * D(T_{i,j,k}) \end{cases} \quad (5.6)$$

where t is the execution time and α is the percentage of time where the SLA violations can be tolerated before the service providers incur any penalty. The value, α is different for each CoS and the higher the priority of CoS, the lower the α value.

The Total Profit for all the tasks, T is given as

$$Profit(T) = \sum_{i=1}^{n} MAX(Rev(T_{i,j,k}, t)) \quad (5.7)$$

### 5.3.2 AQoSSA Components and Algorithms

Fig 5.2 depicts the main components of the AQoSSA. The SLA negotiator module is added to negotiate the SLA agreement between service providers and end users. Once an end user submits a job request with its associated CoS, the service provider estimates the deadline and specifies the penalties in case of violations. The SLA negotiator ensures that an agreement is reached between service providers and end users before a job is accepted.

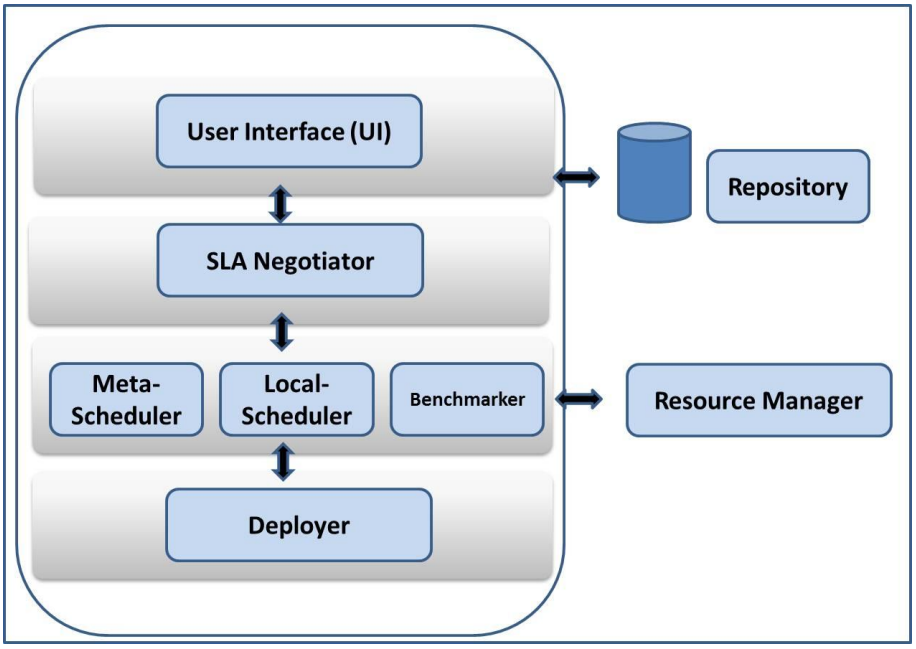


**Figure 5.2: Main components of AQoSSA**

AQoSSA applies the ASA mechanism with dynamic z value at each interval and schedules the task according to the CoS. The Gold service with CoS value equals to one will be scheduled first followed by Silver and Bronze service. AQoSSA is able to reschedule a task with lower priority when the task is waiting in the queue but the deadline is not expired. Rescheduling allows the higher priority task with immediate deadline to execute first. Figure 5.3 shows the proposed AQoSSA.

Input:
$I_i$, a set of different task request at interval i
$p_m$ or $q_n \in S_i$, , where $S_i$ is a set of resources in site i
Output
A schedule of $T_{i,j,k}$ into $p_m$ or $q_n$

// estimate the task length at each interval and sort the tasks according to Class of Service
∀ task request, $I_i$, in waiting queue
Sort the task $T_{i,j,k}$ based on CoS
estimate the execution time, $e_{i,j,k}$
Sort the task $T_{i,j,k}$ in descending order based on the estimated execution time
Generate a sorted task requests list $K_i$

```
Compute mean_i from K_i

∀ servers and clusters in site I  // generate the primary resource list
    Select potential resources and generate sorted primary resource list, P_i

∀ desktop computers in site I  // generate the secondary resource list
    Select available desktop computers and generate sorted secondary resource
    list, Q_i

// compare the multiplication of z and mean with the parameter value to
// determine whether to schedule the task to primary or secondary resource
∀ Task, T_i,j
    if  l_i,j,k > mean_i * z
        allocate T_i,j,k to p_m that return maximum profit
    Otherwise
        allocate T_i,j,k to q_n that return maximum profit

    // reschedule the tasks when CoS is equal to 3 and the deadline is not expired
    if the deadline of the new task is expired
        ∀ tasks in waiting queue with lower priority than new task
            reschedule waiting task to allow new task to meet the deadline

    Repeat the experiment for each interval,
    With the dynamic value, z ± 0.1 to get the maximum number of tasks
     that meet the deadline
```

**Figure 5.3: Proposed AQoSSA**

## 5.4 Experiment Setup

An experimental testbed with the implementation of the same components as Chapter 4 is developed to execute the testing of services in Grid and Cloud environment. An extra component, the SLA negotiator which is used to negotiate the SLA agreement between service providers and end users is added to the testbed. The test cases used to verify the correctness in implementing the proposed AQoSSA, MIN-MIN QoS and MAX-MIN QoS are presented in Appendix A.3.

The resources and parameters used to evaluate the performance of the proposed algorithm are the same set of resources and job parameters used in Chapter 4 and are shown again in Table 5.5. A set of evaluations is carried out to compare the performance of the proposed AQoSSA against MIN-MIN QoS and MAX-MIN QoS. The performance metric used for the comparison is the total number of job completed and the revenue. Several test cases: 1) Random job arrival with random job length; 2) Random job arrival with big gap job length; 3) Random job arrival with job length in Poisson distribution are used for the experiment.

**Table 5.5 A predefined set of resources and job parameters**

| Description | Value Range |
|---|---|
| Number of sites | 3 – 5 |
| Number of hosts per site | 20 – 50 |
| Number of CPU | 1 – 4 |
| Number of core per CPU | 1 – 8 |
| CPU speed | 0.8 – 3.2 GHz |
| Number of tasks | 500 |
| Execution time of task (second) | 1 – 1000 |
| Type of Web Services | 1 – 10 |
| Class of Service (1 – Highest Priority) | 1 – 3 |

## 5.5 Result and Analysis

Table 5.6 shows the reliability (in percentage) for different test cases using the three different algorithms. Figure 5.4 displays the results from Table 5.6 in graphical form. The reliability is computed as:

reliability (%) = (total number of job completed / total number of job) * 100% (5.8)

**Table 5.6 Reliability (%) for different algorithms**

| | MIN-MIN QoS | | | MAX-MIN QoS | | | AQoSSA | | |
|---|---|---|---|---|---|---|---|---|---|
| | Number of Job Completed | Number of Job Violated | Reliability % | Number of Job Completed | Number of Job Violated | Reliability % | Number of Job Completed | Number of Job Violated | Reliability % |
| **Random job arrival and random job length (Benchmarking)** | 464 | 36 | 92.8 | 469 | 31 | 93.8 | 493 | 7 | 98.6 |
| **Random job arrival and big gap job length (Benchmarking)** | 458 | 42 | 91.6 | 465 | 35 | 93.0 | 484 | 16 | 96.8 |
| **Random job arrival and job length in Poisson Distribution (Benchmarking)** | 462 | 38 | 92.4 | 472 | 28 | 94.4 | 490 | 10 | 98.0 |

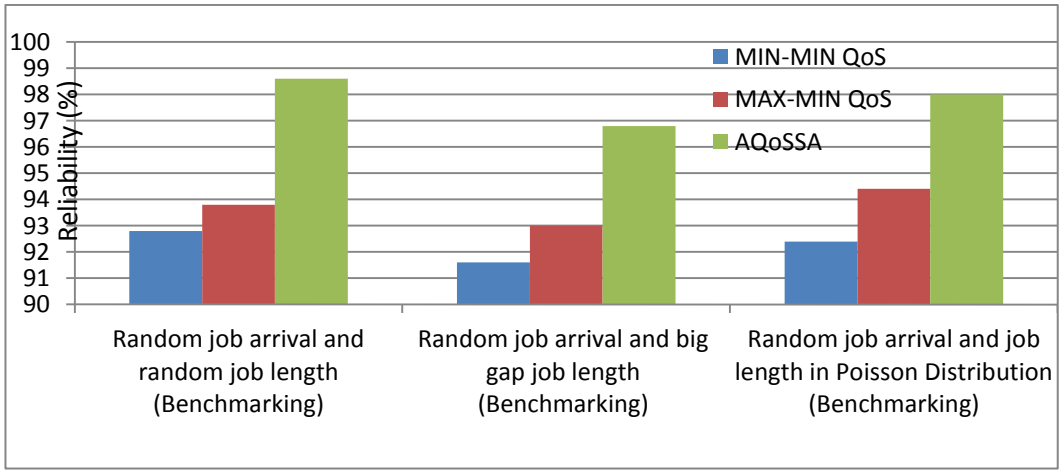


**Figure 5.4: Reliability (%) for different algorithms**

A total of 500 jobs are used for the testing. The results showed that the reliability of MIN-MIN QoS and MAX-MIN QoS scheduling algorithm is around 92% - 94% as shown in Table 5.6. Whereas, the number of jobs completed using AQoSSA is more than 480. This translates to achieving a reliability of more than 96% for AQoSSA.

Table 5.7 tabulates the result of the reliability (in percentage) and the percentage of improvements. The experimental results shown in Table 5.7 inferred that AQoSSA yield the best results as compared to the other scheduling algorithms in terms of reliability. It is observed that the improvement of AQoSSA over MIN-MIN QoS is around 5% in terms of reliability. It is also observed that AQoSSA performs better than MIN-MIN QoS by between 3% - 4% in terms of reliability for different test cases. Hence, AQoSSA provides better scheduling decision in reducing the number of jobs violated the SLA. The implementation of the rescheduling mechanism allows most of the jobs to complete within the deadline.

**Table 5.7 Reliability (%) for different algorithms and percentage of improvement of AQoSSA as compared to other algorithms**

| | MIN-MIN QoS | MAX-MIN QoS | AQoSSA | % improvement of AQoSSA over MIN-MIN (QoS) | % improvement of AQoSSA over MAX-MIN (QoS) |
|---|---|---|---|---|---|
| Random job arrival and random job length (Benchmarking) | 92.8 | 93.8 | 98.6 | 5.8 | 4.8 |
| Random job arrival and big gap job length (Benchmarking) | 91.6 | 93.0 | 96.8 | 5.2 | 3.8 |
| Random job arrival and job length in Poisson Distribution (Benchmarking) | 92.4 | 94.4 | 98.0 | 5.6 | 3.6 |

Table 5.8 shows the total profit (unit) for different experimental cases using the different algorithms. The total profit is computed using the equation 5.7. It is observed that from Table 5.8, the total penalty of the MIN-MIN QoS scheduling algorithm is the highest when compared to MAX-MIN QoS scheduling algorithm and AQoSSA. AQOSSA achieves the highest revenue and lowest penalty when compared to MIN-MIN QoS and MAX-MIN QoS

scheduling algorithm because the total number of job completed within the deadline is the highest and the total number of job violated the SLA is the lowest.

**Table 5.8 Total profit (unit) for different algorithms**

| | MIN-MIN QoS | | | MAX-MIN QoS | | | AQoSSA | | |
|---|---|---|---|---|---|---|---|---|---|
| | Total Revenue | Total Penalty | Total Profit | Total Revenue | Total Penalty | Total Profit | Total Revenue | Total Penalty | Total Profit |
| **Random job arrival and random job length (Benchmarking)** | 13511 | 1145 | 12366 | 15764 | 910 | 14854 | 17311 | 150 | 17161 |
| **Random job arrival and big gap job length (Benchmarking)** | 16775 | 1975 | 14800 | 20584 | 1890 | 18694 | 22175 | 390 | 21785 |
| **Random job arrival and job length in Poisson Distribution (Benchmarking)** | 9858 | 1055 | 8803 | 11950 | 1055 | 10895 | 12204 | 135 | 12069 |

Figure 5.5 displays the results from Table 5.8 in a graphical form and Table 5.9 shows the profit and the percentage of improvement for different experimental cases using the three algorithms. The experimental results in Table 5.9 show that AQoSSA outperforms MIN-MIN QoS and MAX-MIN QoS by a significant margin for different test cases in terms of profitability. Compared to other approaches, AQoSSA makes between 10% - 47% improvements as AQoSSA is able to maximize the number of jobs completed while minimizing the number of occurrences of jobs that fail to adhere to the SLA. In other words, AQoSSA maximizes the revenue gain and minimizes the total cost penalty. In summary, AQoSSA is able to maximize the reliability and the total profit while meeting the end users QoS requirements.

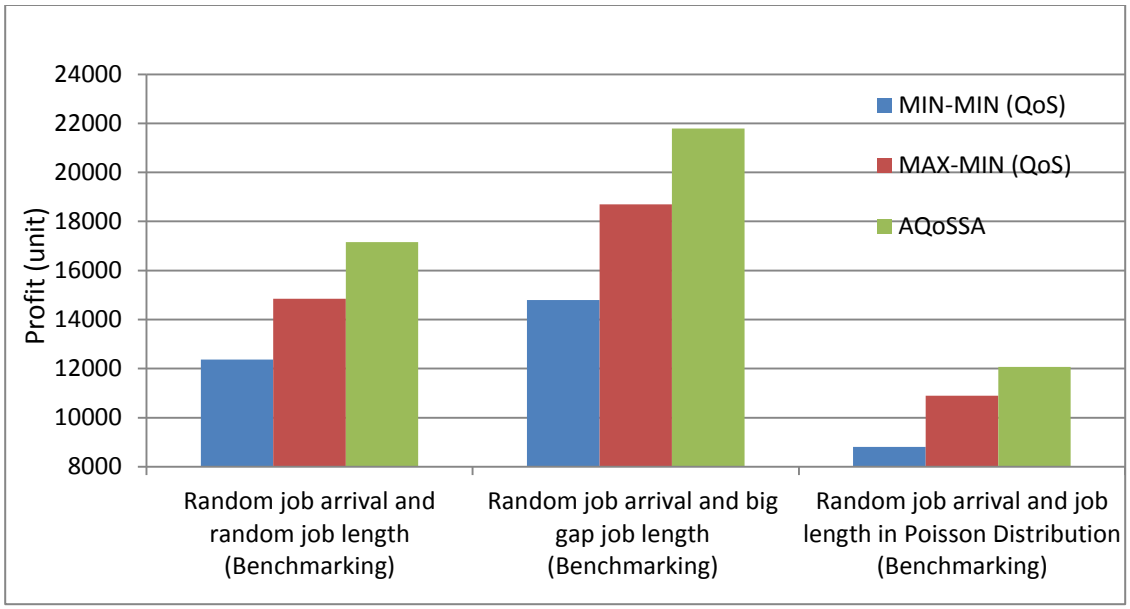


**Figure 5.5: Total profit of different algorithms**

**Table 5.9 Total profit of different algorithms and percentage of improvement of AQoSSA as compared to other algorithms**

| | MIN-MIN (QoS) | MAX-MIN (QoS) | AQoSSA | % improvement of AQoSSA over MIN-MIN (QoS) | % improvement of AQoSSA over MAX-MIN (QoS) |
|---|---|---|---|---|---|
| Random job arrival and random job length (Benchmarking) | 12366 | 14854 | 17161 | 38.8 | 15.5 |
| Random job arrival and big gap job length (Benchmarking) | 14800 | 18694 | 21785 | 47.2 | 16.5 |
| Random job arrival and job length in Poisson Distribution (Benchmarking) | 8803 | 10895 | 12069 | 37.1 | 10.8 |

## 5.6 Chapter Summary

This chapter presented the solution that guarantees end users QoS requirements and provides cost management for service providers in Grid and Cloud environment. AQoSSA is proposed as the enhancement of ASA to support users QoS requirements. In AQoSSA,

the resources are divided into groups using k-mean and then the revenue obtained from leasing the resources is computed from the mean CPU benchmark for each group. The tasks are sorted according to the CoS and the adaptive mechanism is applied to the scheduling decision. AQoSSA minimizes the total number of jobs that violate the SLA by using a rescheduling mechanism.

When AQoSSA is compared to scheduling algorithms that utilizes MIN-MIN QoS and MAX-MIN QoS, there is a 3% - 6% improvement in reliability. In terms of overall profitability, the implementation of AQoSSA yielded a significant improvement of 10% - 47% over the other two approaches.

In summary, AQoSSA has significantly outperformed MIN-MIN QoS and MAX-MIN QoS algorithms in reliability and profitability while guaranteeing the end users QoS requirements.

# CHAPTER 6 CONCLUSIONS AND FUTURE DIRECTIONS

This chapter summarizes this thesis and provides suggestions for future research. The first section details the summary of this thesis contribution and discusses on some of the related technical matters. This is followed by a discussion on future work.

This thesis began with the investigation of the underlying concepts, characteristics, system architectures and problems in Grid and Cloud Computing. Next, the Web Services and Service Oriented Architecture (SOA) are discussed. Subsequently, the job scheduling process and the existing job scheduling algorithms for Grid and Cloud Computing are explored.

Following the studies, a Hybrid Scheduling Algorithm (HSA) with automatic deployment mechanism is proposed to maximize resources utilization and minimize the makespan in the Grid and Cloud environment. Next, the Adaptive Scheduling Algorithm (ASA) with benchmarking is proposed. ASA enhances HSA through the use of benchmarking as well as the use of an adaptive mechanism. ASA is able to provide a more dynamic scheduling decision making process as compared to HSA. Finally, the Adaptive QoS Scheduling Algorithm (AQoSSA) with rescheduling mechanism is proposed to enhance the ASA to meet users QoS requirements. AQoSSA is able to maximize reliability and profit while guaranteeing the users QoS requirements.

In order to evaluate the proposed algorithms and mechanisms, a Grid and Cloud testbed is developed. The experimental results show that the proposed algorithms maximize resources

utilization and minimize makespan in Grid and Cloud environment. Furthermore, the proposed AQoSSA also maximizes reliability and profit while guaranteeing the users QoS requirements.

## 6.1 Thesis Contributions

In this thesis, the following three novel algorithms have been proposed and developed:

- Hybrid Scheduling Algorithm (HSA) with automatic deployment mechanism.
- Adaptive Scheduling Algorithm (ASA) with benchmarking.
- Adaptive Quality of Service Scheduling Algorithm (AQoSSA) with rescheduling mechanism.

All the proposed algorithms and mechanisms are evaluated via the Grid and Cloud testbed using different job length and job arrival distributions. In general, the experimental results show that these algorithms and mechanisms made significant improvements in arriving at the job scheduling decisions.

### 6.1.1 HSA with Automatic Deployment

The proposed HSA is running on a hierarchical structure where the system consists of one centralize meta-scheduler and multiple distributed local schedulers. The dynamic scheduling is used at the centralized meta-scheduler to support on-line mode while the static scheduling is implemented at multiple distributed local schedulers for off-line mode. In addition, a backup meta-scheduler is implemented to overcome the potential of a single failure in a centralized meta-scheduler. HSA implements the automatic deployment mechanism to automate the process of services deployment to the resources. HSA uses a

fix threshold value, z, to make scheduling decision either to dispatch task to primary or secondary resources. Experimental results show that the HSA and automatic deployment mechanism minimize the makespan by 1% – 11% as compared to MIN-MIN and MAX-MIN scheduling algorithms. In summary, HSA is able to maximize resources utilization and minimizes makespan in Grid and Cloud environment.

### 6.1.2 ASA with Benchmarking

The HSA is further improved by incorporating benchmarking and adaptive mechanism. ASA, the enhancement of HSA uses a combination of resource benchmark and application benchmark to estimate the job length. Besides, ASA computes the mean value and uses a dynamic threshold value, z, at each interval to make scheduling decision to dispatch task either to primary or secondary resources. ASA is able to adapt to different experimental settings since the value z is dynamic. The experimental results show that ASA with benchmarking minimize the makespan by 5% – 17% as compared to MIN-MIN and MAX-MIN scheduling algorithms with benchmarking. ASA enhances the HSA by 1% - 4% in the area of minimizing the makespan. In summary, ASA is able to maximize resources utilization and minimizes makespan in Grid and Cloud environment.

### 6.1.3 AQoSSA with Rescheduling

AQoSSA is the enhancement of the ASA to maximize reliability and profit while satisfying the users QoS requirements. AQoSSA verifies the users QoS requirements such as Class of Service (CoS) and deadline specified in the SLA and runs the rescheduling mechanism when necessary. Experimental results show that AQoSSA maximizes the reliability by 3% - 6% and increases the total profit margin by 10% - 47% over the commonly use MIN-MIN

QoS and MAX-MIN QoS scheduling algorithms. In summary, the use of AQoSSA made significant contribution to the performance improvements with its ability to maximize reliability and profit while guaranteeing the users QoS requirements.

### 6.1.4 Grid and Cloud Testbed

A Grid and Cloud testbed is developed to evaluate the proposed algorithms and mechanisms. The testbed implements all the modules of the proposed algorithms. The components of the testbed include the User Interface (UI), meta-scheduler, local-scheduler, resource manager, benchmarker, deployer and the SLA negotiator.

The UI supports batch job submission at each interval. The meta-scheduler schedules jobs to multiple sites while the local scheduler schedules the tasks at each site according to each locally available individual resource. A resource manager provides the resource information for each site.

The deployer is developed to provide the automatic deployment of Web Services to the resources without pre-installed Web server. The automatic deployment mechanism is implemented on .NET and J2EE environment. The benchmarker component estimates a job length and provides the micro-benchmark that used to profile resources capabilities and the application-specific benchmark that measures the performance of resources when executing different types of jobs. Both the micro-benchmark and application benchmark provide the estimation of job length.

The SLA negotiator is developed to negotiate the SLA agreement between service providers and end users. The SLA negotiator ensures that an agreement is reached between service providers and end users before a job is accepted.

Once the Grid and Cloud testbed has been developed, the proposed algorithms are evaluated using different job length and job arrival distributions. Various configurations are used in the setup and the testing results can be stored for future analysis.

### 6.1.5 Summary

In summary, this thesis has contributed in several technological areas. Firstly, the job scheduling process in Grid and Cloud Computing are explored. The various types of existing scheduling algorithms are studied and the problems related to the job scheduling process are identified.

Secondly, three novel job scheduling algorithms in Grid and Cloud environment have been proposed and implemented. These algorithms are capable of maximizing the resources utilization and minimize makespan in the Grid and Cloud environment. In addition, these algorithms are shown to be able to maximize reliability and profit while guaranteeing the users QoS requirements.

Finally, a Grid and Cloud testbed is implemented. The Grid and Cloud testbed is used as the simulation platform to evaluate the performance of the proposed algorithms. Different setup configurations are used and the results are then thoroughly analysed. The experimental results show that the HSA and ASA outperform MIN-MIN and MAX-MIN

by between 1% - 10% and 5% - 17% respectively in terms of makespan. AQoSSA outperforms MIN-MIN QoS and MAX-MIN QoS by between 3% - 6% in terms of reliability and 10% - 47% in terms of profit while guaranteeing the QoS requirements.

## 6.2 Suggestions for Future Work

The growing interests in Grid and Cloud Computing have led to many new approaches to manage the architecture. There are many issues related to this new computing environment which open up new areas for further research. One of the major issues is the energy aware resource allocation.

As the number of computing resources increase every year, the energy consumption used for computation is climbing. This increasing energy consumption has contributed to the pace of global warming. With the environmental concerns, Green Computing is gaining much popularity and has attracted many research studies. Green computing refers to environmentally sustainable computing which studies and practices virtually all computing efficiently and effectively with little or no impact on the environment (Lo and Qian, 2010).

Energy aware resource allocation is one of the mechanisms that have the ability to reduce energy consumption in computing resources. Usually, in existing data centres, there are many servers used for different services but the resources are operating at low utilization levels. The under-utilized hardware continues to contribute to the energy consumption. With the rising cost of energy, the energy aware resource allocation mechanism can be proposed to provide power savings in the Grid and Cloud environment.

In summary, Grid and Cloud Computing have emerged as the new paradigm for the provisioning of computing resources. The growing interests in these technologies will create a lot of research opportunities and the benefits of Grid and Cloud will become more apparent.