2013

Designing and Handling Failure issues in a Structured Overlay Network Based Grid

Amar Bahadur Patel
Universty of Windsor

Designing and Handling Failure issues in a Structured Overlay Network Based Grid

by

Amar Bahadur Patel

A Thesis
Submitted to the Faculty of Graduate Studies
through the **School of Computer Science**
in Partial Fulfillment of the Requirements for
the Degree of **Master of Science** at the
University of Windsor

Windsor, Ontario, Canada

2013

Handling of Failure issues in a Structured Overlay Network Based Grid

by

Amar Bahadur Patel

APPROVED BY:

_____

External Reader
Dr. Kemal Tepe
Department of Electrical and Computer Engineering

_____

Internal Reader
Dr. Arunita Jaekel
School of Computer Science

_____

Advisor
Dr. Robert D. Kent
School of Computer Science

_____

Chair of Defense
Dr. Subir Bandyopadyhay
School of Computer Science

August 28, 2013

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Grid computing is the computing paradigm that is concerned with coordinated resource sharing and problem solving in dynamic, autonomous multi-institutional virtual organizations. Data exchange and service allocation between virtual organizations are challenging problems in the field of Grid computing, due to the decentralization of Grid systems. The resource management in a Grid system ensures efficiency and usability. The required efficiency and usability of Grid systems can be achieved by building a decentralized multi-virtual Grid system.

In this thesis we present a decentralized multi-virtual resource management framework in which the system is divided into virtual organizations, each controlled by a broker. An overlay network of brokers is responsible for global resource management and managing the allocation of services. We address two main issues for both local and global resource management: 1) decentralized allocation of tasks to suitable nodes to achieve both local and global load balancing; and 2) handling of both regular and broker failures. Experimental results verify that the system achieves dependable performance with various loads of services and broker failures.

DEDICATION

To my parents, and my loving wife Seema

ACKNOWLEDGEMENTS

I would first like to express my sincere gratitude and appreciation to Dr. Robert Kent, my supervisor, for his support and guidance throughout the course of this research.

My sincere appreciation goes to my parents for their unconditional support, and also to my wife Seema for her encouragement and support in all these years.

I would also like to thank Dr. Arunita Jaekel and Dr. Kamal Tepe for serving on my thesis committee.

Lastly, I would like to thank all my colleagues in the Computer Science department who helped me through their inputs to improve my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Introduction to research topic area

A distributed network computing system [1] is a virtual computer formed by a networked set of heterogeneous machines that agree to share their local resources with each other. Distributed Computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony. In such a model, users access services based on their requirements without regard to where the services are hosted or how they are delivered.

Several computing paradigms [2] have promised to deliver this utility computing vision that includes Cluster computing, Grid computing and, more recently, Cloud computing. The latter term denotes the infrastructure as a somewhat diffuse and transparent entity, hence "Cloud", by which businesses and users are able to access applications from anywhere in the world on demand, using virtualization of management. Thus, the computing world is rapidly transforming towards developing software for millions to consume as a service rather than to run on their individual computers.

1.1.1 Grid Computing

Grid computing is the computing paradigm that is concerned with "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations" [1, 2]. A virtual organization (VO) can be defined as a collection of computing nodes in which

each participating node can acquire or provide services from/to other nodes inside/outside the organization [3]. Grid computing uses middleware

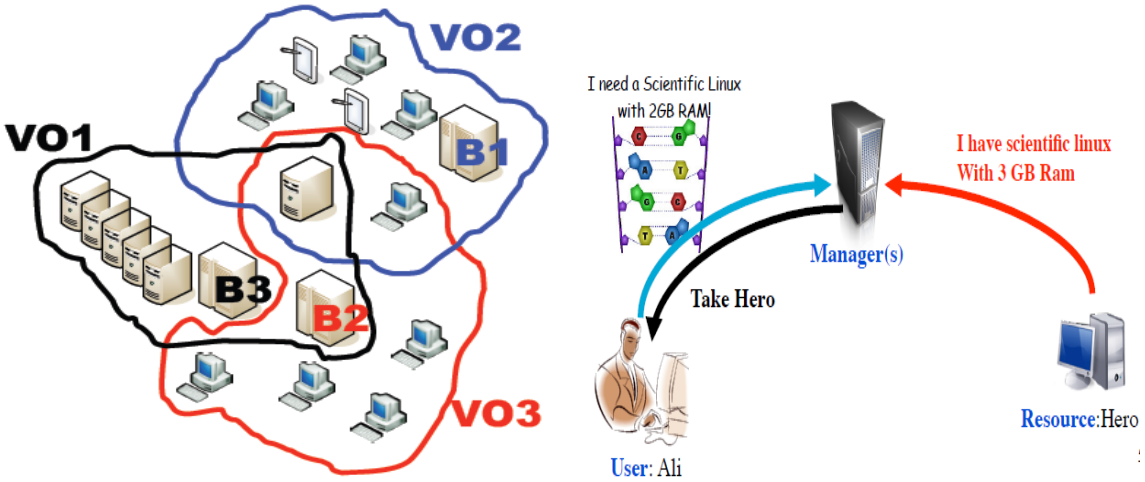


Figure 1. Illustrating the Grid computing paradigm with an example of User Management-Resource interaction.

to coordinate disparate IT resources across a network, allowing them to function as a virtual whole. Grids address two distinct but related goals: providing remote access to IT assets, and aggregating processing power. The most obvious resource included in a grid is a processor, but grids also encompass sensors, data-storage systems, applications, and other resources. The Grid computing paradigm with an example of user management-resource interaction are depicted in Figure 1.

The grid provides a series of distributed computing resources through LAN or WAN. It is using a super virtual computer as terminal user of the application. This idea will not only realize safe and coordinate resource sharing among persons, organizations, and resources, but also create a virtual and dynamic organization. Grid computing is a method of distributed computing. It includes location and organization, software, and hardware to provide unlimited power. Each source can cooperate and access others, but

cloud computing is better; it has many advantages over grid computing in many ways. Cloud computing evolves from grid computing and provides on-demand resource provisioning. Grid computing may or may not be in the cloud depending on what type of users who are using it. If the users are systems administrators and integrators, what they care is how things are maintained in the cloud. They upgrade, install, and virtualize servers and applications. If the users are consumers, they do not care how things are running in the system. Grid computing requires the use of software that can divide and form the pieces of a program from different resources as one large system image . One concern about the grid is that if one piece of the software on a node fails, then other pieces of the software on other nodes may also fail. This is alleviated if that component has a failover component on another node, but problems can still arise if components rely on other pieces of software to accomplish one or more grid computing tasks. Large system images and associated hardware to operate and maintain them can contribute to large capital and operating expenses [4].

1.1.1.1 Brief Description of Resource Management Concepts in Grid

A grid-computing system makes use of computer resources from multiple administrative domains that are applied collectively to solve a problem that has demanding requirements such as a large amount of processing power, storage space, bandwidth and so on. However, a more generalized and formal definition of a Grid system can be described as "a large-scale, geographically distributed, hardware and software infrastructure composed of heterogeneous networked resources owned and shared by multiple administrative organizations, which are coordinated to provide transparent, dependable, pervasive, and consistent computing support to a wide range of applications. These applications can

perform distributed computing, high throughput computing, on-demand computing, data-intensive computing, collaborative computing or multimedia computing"[5].

From the definitions above, it can be clearly seen that the base of grid technology is the concept of resource sharing/management. The term resource management in grid computing can be defined as those operations that control the way that grid resources and services are made available for use by entities like users, applications, and services [6] to ensure efficient utilization of computer resources and for optimal performance of specific tasks. Due to the complexity, heterogeneity, and the dynamic nature of grid computing environments, resource management is faced with challenges making it a complex task to match the capabilities of available resources to the needs of the entities listed above [6, 7]. Some grid resource classifications are storage resources, network resources, and computational resources which have capabilities like storage capacity available on disk, bandwidth of the network, and processor speed [8]. To handle the wide variety of the software applications and hardware's used in grid environments over different forms of grid networks, software known as middleware is used. One of the most important components of the middleware is the resource manager which handles resource selection and job scheduling [9, 10].

The discovery of resources becomes more difficult when resources owned by many organizations having different resource management policies and cost models are distributed over a wide geographic area and are heterogeneous in nature. To handle these problems, a number of different scheduling and resource management algorithms and methods have been developed and implemented.

Resource Manager as Middleware: Middleware, which does not have a unanimous definition [11], can generally be described as a layer of software that handles the heterogeneous functions of a distributed system like in grids or clouds. It exists between the application and the underlying components such as the operating system, networks, and hardware connecting them together and creating a useful environment. There are a number of different grid middleware kits such as Globus toolkit, ARC, gLite, UNICORE etc.

Resource Discovery: As one of the key issues in a grid system is how to manage all types of distributed resources for job executions. A mechanism should be provided by the grid infrastructure to discover the relevant resource for its corresponding requests. In relation to this [12], pointed that one of the main capabilities of a grid infrastructure is the need to support a resource discovery mechanism which in turn plays an important role in the management of grid resources. Resource discovery mechanisms in grid systems, which make use of trust relationship between resource requesters and providers prevent malicious attempts in the environment and also reduce extra overhead in the existing complex grid systems. An efficient trust based resource discovery mechanism ensures a safer environment for communication; enhance the quality of service (QoS) and also easier decision making for allocating resources between the requesters and providers.

Resource Allocation and Scheduling: Grid resources are distributed heterogeneously over a large geographical area to numerous users simultaneously. To manage the use of these resources and others properly, tasks need to be scheduled precisely and allocated the corresponding demanded resources accordingly. Grid is a dynamic system, which changes continuously with time as a result of a number of factors

such as availability of new resources, system/resource failure, new requests, completion of an executing task etc. When any of these occurs, there is a need for jobs to be rescheduled or re-allocation of resources for execution.

1.1.2 Cloud Computing

Cloud computing provides a large-scale distributed computing paradigm that is driven by economics of scale [13, 14], in which a pool of abstracted, virtualized, dynamic, scalable, managed computing power, storage, platform, and services are delivered on demand to external customers over the Internet. The Cloud computing paradigm with an example of user management- resource interaction are depicted in Figure 2.

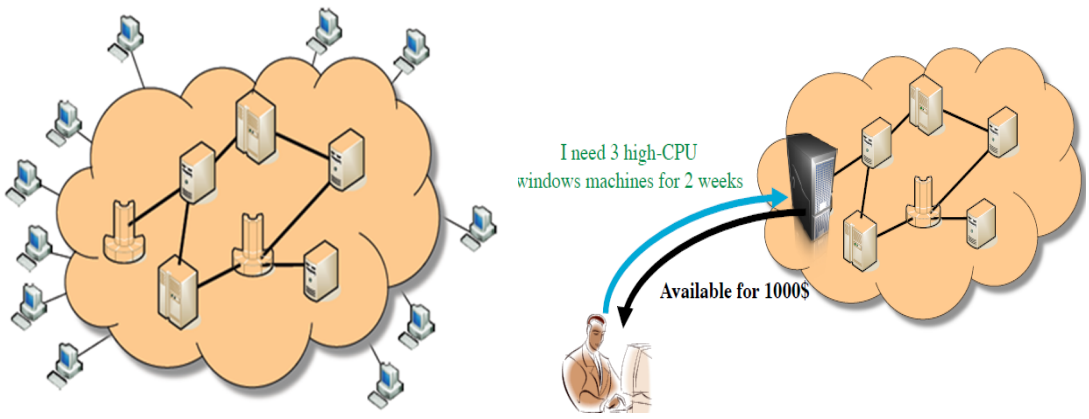


Figure 2. Illustrating the Cloud computing paradigm with an example of User Management-Resource interaction.

Following are a few key points for defining cloud computing. 1) it is massively scalable,which driven by the economics of scale [13, 14], 2) it can be encapsulated as an abstract entity that delivers different levels of services to customers outside the Cloud, 3) the services can be dynamically configured (via virtualization or other approaches) and delivered on demand.

Governments, research institutes, and industry leaders have rushed to adopt Cloud Computing in order to solve their ever increasing computing and storage problems arising in the Internet Age. There are three main factors contributing to the surge and interests in Cloud Computing: 1) rapid decrease in hardware cost and increase in computing power and storage capacity, and the advent of multi-core architecture and modern supercomputers consisting of hundreds of thousands of cores; 2) the exponentially growing data size in scientific instrumentation/simulation and Internet publishing and archiving, and 3) the wide-spread adoption of Services of Computing and Web applications

1.1.2.1 Cloud Computing Economics

Based on recent work by Haas et al [15] and Sharma et al [16], we make the following. Observations about Cloud Computing economic models: 1) In deciding whether hosting a service in the cloud makes sense over the long term, the fine grained economic models enabled by Cloud Computing make tradeoff decisions more fluid and, in particular, the elasticity offered by clouds serves to transfer risk as well. Although hardware resource costs continue to decline, they do so at varying rates; for example, computing and storage costs are falling faster than WAN costs. Cloud Computing can track these changes and potentially pass any derived benefits through to the customer more effectively than building one's own data center, resulting in a closer match of expenditure to actual resource usage. 2) In making the decision about whether to move an existing service to the cloud, one must additionally examine the expected average and peak resource utilization, especially if the application may have highly variable spikes in resource demand.This places practical limits on real-world utilization of purchased equipment and

various operational costs that vary depending on the type of cloud environment being considered.

1.1.3 The comparison between Cloud Computing and Grid Computing

First, we can compare those from job scheduling of grid computing. Job scheduling is the core value and aim of grid technology, its aim is to use all kinds of resources. It can divide a huge task into a lot of independent and no related sub tasks, and then let every node do the jobs. Even any node fails and doesn't return a result, it doesn't matter; the whole process will not be affected. Even one node crashes, the task should be reassigned to other nodes. Just like grid computing, cloud computing will make a huge resource pool through grouping all the resources. But the resource provided by the cloud is to complete a special task. For example, a user may apply resource from the resource pool to deploy its application, not submit its task to the grid and let grid complete it [17]. From this point, the construction of the grid is to complete a specified task, and then there will be biology grid, geography grid, and national educational grid and so on. Cloud computing is designed to meet general application requirement.

Second comparison, the cloud will have effects in three aspects: the application in internet, product application model, and IT product development direction [18]. Of course, this change is not subversion but some new characters that have been added. This advantage is a challenge to grid technology. When grid come it into being, it has some advantages, such as: you can provide unlimited compute power through any computer, and can get a great deal of information. This environment can help an enterprise to complete tasks that are very hard before use their systems efficiently to meet the user's requirement and decrease the management cost. Cloud computing extends these

advantages. More and more applications will be completed through the internet by cloud computing. Cloud computing will extend the application of hardware and software, and will change the application model of hardware and software. Users can get an application environment or the application itself not buying new servers and new software. To the users, the hardware or the software need not on his side or only used by himself, it can be available and virtual resources. And available resources are not limited inside the enterprise, it can be extended hardware and software attained through the internet.

The development direction of IT product will be changed to meet the above two conditions. Cloud computing provides services in the following ways: 1) SAAS (Software as a service). This kind of cloud computing transfer programs to many users through a web browser. In the user's eyes, he/she can save a lot of money in the servers and the authorization of software. In the supporters' eyes, he needs to maintain only one program, which will also save costs [19]. 2) Utility Computing. This is an old idea; but, it is used by Amazon, IBM, Sun and other companies that provide storage services and virtual servers in recent years. This cloud computing creates virtual data centers for IT business to collect memory, IO device, storage and computing power to make a huge virtual resource to serve the whole net [20]. 3) Network service. It has a close relation with SAAS, network service providers provide API to make developers develop more applications based on the internet, not only the PC program. 4) Platform as a service. This kind of cloud computing provides a development environment. You can use the middleman's device to develop your own program and deliver it to users through the internet and servers. 5) Management Service Provider. It is one of the oldest application in cloud computing. This kind of application faces IT business not terminal users, it is

often used virus scanning and program monitoring. 6) Business Service Platform. It is the mixture of SAAS and MSP. This kind of cloud computing provides a platform for the interaction between users and providers, such as personal budget management system, it can manage his budget and coordinate all the services he has booked according to the user's setup. 7) Integration of internet. It is to integrate all the companies that are doing the similar jobs. That will make it easier for users to choose and select the service providers. 8) Infrastructure as a Service and more a consumer can get the service from a full computer infrastructure through the Internet [21]. This type of service is called Infrastructure as a Service (IaaS). Internet-based services such as storage and databases are part of the IaaS. Other types of services on the Internet are Platform as a Service (PaaS) and Software as a Service (SaaS). PaaS offers full or partial application development that users can access, while SaaS provides a complete turnkey application, such as Enterprise Resource Management through the Internet. The IaaS divides into two types of usage: public and private. Amazon EC2 uses public server pools in the infrastructure cloud [22]. A more private cloud service uses groups of public or private server pools from an internal corporate data center. You can use both types to develop software within the environment of the corporate data center with EC2, temporarily extend resources at low cost say for testing purposes. The mix may provide a faster way of developing applications and services with shorter development and testing cycles. To grid computing, though its resources have been pooled, it looks like a huge resource pool from outside. But to the user who has submitted a special task, he doesn't know which node will complete his job. What he need do is to submit his job to the grid according to a special style, and then what he will do next is waiting for the result. And the grid job

schedule system will look for the resource that is matched to job, and find idle physical node, send out the job until the job will be finished. Though grid can realize parallel job processing, the user has to prepare the algorithm himself, and send them to different physical nodes. This process is a little complicated, that is why many grid computing is built to complete special requirements. Cloud computing will cut the physical resource through virtual method. From this point it can realize allocate resource according to the need and increase automatically. This kind of increase can't exceed the up limit of the physical nodes. Though from the view of a control point, the cloud will look all the IT resources as a resource pool, different physical node will be divided within different resource pool. That is the difference between grid computing and cloud computing in the allocation of resource. From the concept, cloud computing is actually a kind of distributed computing; this computing model has advantages and huge potential over the compute model in traditional databases. At the same time, some experts have pointed out that automation technology is the base of any cloud computing infrastructure. Automation technology is also the base of any advanced computing technology. If you want to use cloud computing in any case, it means you have no reusable process, and it also means you are trying to let others do the job that you are not qualified to do.

1.1.4 Resource management in both Grid and Cloud Computing

This section describes the resource management found in Grids and Clouds, covering topics such as the compute model, data model, virtualization, and monitoring. These topics are extremely important to understand the main challenges that both Grids and Clouds face today, and will have to overcome in the future.

Compute Model: Most Grids use a batch-scheduled compute model, in which a local resource manager (LRM), such as PBS, Condor, SGE manages the compute resources for a Grid site and users submit batch jobs (via GRAM) to request some resources for some time. Many Grids have policies in place that enforce these batch jobs to identify the user and credentials under which the job will run for accounting and security purposes, the number of processors needed, and the duration of the allocation. For example, a job could say, stage in the input data from a URL to the local storage, run the application for 60 minutes on 100 processors, and stage out the results to some FTP server. The job would wait in the LRM's wait queue until the 100 processors were available for 60 minutes, at which point the 100 processors would be allocated and dedicated to the application for the duration of the job. Due to the expensive scheduling decisions, data staging in and out, and potentially long queue times, many Grids don't natively support interactive applications; although there are efforts in the Grid community to enable lower latencies to resources via multi-level scheduling, to allow applications with many short-running tasks to execute efficiently on Grids [23]. Cloud Computing compute model will likely look very different, with resources in the Cloud being shared by all users at the same time (in contrast to dedicated resources governed by a queuing system). This should allow latency sensitive applications to operate natively on Clouds, although ensuring a good enough level of QoS is being delivered to the end users will not be trivial, and will likely be one of the major challenges for Cloud Computing as the Clouds grow in scale, and number of users.

Data Model: While some people boldly predict that future Internet Computing will be towards Cloud Computing centralized, in which storage, computing, and all kinds

of other resources will mainly be provisioned by the Cloud. The Internet Computing will be centralized around Data, Clouding Computing, as well as Client Computing. Cloud Computing and Client Computing will coexist and evolve hand in hand while data management (mapping, partitioning, querying, movement, caching, replication, etc.) will become more and more important for both Cloud Computing and Client Computing with the increase of data-intensive applications. The critical role of Cloud Computing goes without saying, but the importance of Client Computing cannot be overlooked either for several reasons: 1) For security reasons, people might not be willing to run mission-critical applications on the Cloud and send sensitive data to the Cloud for processing and storage, 2) Users want to get their things done even when the Internet and Cloud are down, or the network communication is slow, 3) With the advances of multi-core technology, the coming decade will bring the possibilities of having a desktop supercomputer with 100s to 1000s of hardware threads/cores. Furthermore, many end-users will have various hardware driven end-functionalities, such as visualization and multimedia playback, which will typically run locally. The importance of data has caught the attention of the Grid community for the past decade; Data Grids [24] have been specifically designed to tackle data intensive applications in Grid environments, with the concept of virtual data [25] playing a crucial role. Virtual data capture the relationship between data, programs and computations and prescribes various abstractions that a data grid can provide: location transparency where data can be requested without regard to data location, a distributed metadata catalog is engaged to keep track of the locations of each piece of data (along with its replicas) across grid sites, and privacy and access control are enforced; materialization transparency: data can be either recomputed on the

fly or transferred upon request, depending on the availability of the data and the cost to re-compute. There is also representation transparency where data can be consumed and produced no matter what their actual physical formats and store are, data are mapped into some abstract structural representation and manipulated in that way.

Data Locality: As CPU cycles become cheaper and data sets double in size every year, the main challenge for efficient scaling of applications is the location of the data relative to the available computational resources – moving the data repeatedly to distant CPUs is becoming the bottleneck. [26] There are large differences in IO speeds from local disk storage to wide area networks, which can drastically affect application performance. To achieve good scalability at Internet scales for Clouds, Grids, and their applications, data must be distributed over many computers, and computations must be steered towards the best place to execute in order to minimize the communication costs [26]. Google Map Reduce [27] system runs on top of the Google File System, within which data is loaded, partitioned into chunks, and each chunk replicated. Thus, data processing is collocated with data storage: when a file needs to be processed, the job scheduler consults a storage metadata service to get the host node for each chunk, and then schedules a "map" process on that node so that data locality is exploited efficiently. In Grids, data storage usually relies on a shared file system (e.g. NFS, GPFS, PVFS, Luster), where data locality cannot be easily applied. One approach is to improve schedulers to be data-aware, and to be able to leverage data locality information when scheduling computational tasks; this approach has shown to improve job turnaround time significantly [28].

Combining computer and data management: Even more critical is the combination of the computer and data resource management, which leverages data locality in access patterns to minimize the amount of data movement and improve end application performance and scalability. Attempting to address the storage and computational problems separately forces much data movement between computational and storage resources, which will not scale to tomorrow's peta-scale datasets and millions of processors, and will yield significant underutilization of the raw resources. It is important to schedule computational tasks close to the data, and to understand the costs of moving the work as opposed to moving the data. Data-aware schedulers and dispersing data close to the processors are critical in achieving good scalability and performance. Finally, as the number of processors-cores is increasing (the largest now a days supercomputers have over 200K processors and Grids surpassing 100K processors), there is an ever-growing emphasis for support of high throughput computing with high sustainable dispatch and execution rates. We believe that data management architectures are important to ensure that the data management implementations scale to the required data set sizes in the number of files, objects, and dataset disk space usage while at the same time, ensuring that data element information can be retrieved fast and efficiently. Grids have been making progress in combining computing and data management with data-aware schedulers [28], but we believe that Clouds will face significant challenges in handling data-intensive applications without serious efforts invested in harnessing the data locality of application access patterns. Although data-intensive applications may not be typical applications that Clouds deal with today, as the scales of Clouds grow, it may just be a matter of time for many Clouds.

Virtualization: Virtualization has become an indispensable ingredient for almost every cloud; the most obvious reasons are for abstraction and encapsulation. Just like threads were introduced to provide users the "illusion" as if the computer were running all the threads simultaneously, and each thread was using all the available resources, Clouds need to run multiple (or even up to thousands or millions of) user applications, and all the applications appear to the users as if they were running simultaneously and could use all the available resources in the Cloud. Virtualization provides the necessary abstraction such that the underlying fabric (raw compute, storage, network resources) can be unified as a pool of resources and resource overlays (e.g. Data storage services, Web hosting environments) can be built on top of them. Virtualization also enables each application to be encapsulated such that they can be configured, deployed, started, migrated, suspended, resumed, stopped, etc., And thus provides better security, manageability, and isolation.

There are also many other reasons that Clouds tend to adopt virtualization: 1) server and application consolidation, as multiple applications can be run on the same server, resources can be utilized more efficiently; 2) configurability, as the resource requirements for various applications could differ significantly, some require large storage, some compute, in order to dynamically configure and bundle (aggregate) resources for various needs, virtualization is necessary as this is not achievable at the hardware level; 3) increased application availability, virtualization allows quick recovery from unplanned outages as virtual environments can be backed up and migrated with no interruption in service; 4) improved responsiveness: resource provisioning, monitoring and maintenance can be automated, and common resources can be cached and reused. All

these features of virtualization provide the basis for Clouds to meet stringent SLA (Service Level Agreement) requirements in a business setting, which cannot be easily achieved with a non-virtualized environment in a cost-effective manner as systems would have to be over provisioned to handle peak load and waste resources in idle periods. After all, a virtualization infrastructure can be just thought as a mapping from IT resources to business needs. Grids do not rely on virtualization as much as Clouds do, but that might be more due to policy and having each individual organization maintain full control of their resources (i.e. by not virtualizing them). However, there are efforts in Grids to use virtualization as well, such as Nimbus [29] (previously known as the Virtual Workspace Service [30]), which provide the same abstraction and dynamic deployment capabilities. A virtual workspace is an execution environment that can be deployed dynamically and securely on the Grid. Nimbus provides two levels of guarantees: 1) quality of life: users get exactly the (software) environment they need, and 2) quality of service: provision and guarantee all the resources the workspace needs to function correctly (CPU, memory, disk, bandwidth, availability), allowing for dynamic renegotiation to reflect changing requirements and conditions. In addition, Nimbus can also provision a virtual cluster for Grid applications (e.g. A batch scheduler, or a workflow system), which is also dynamically configurable, a growing trend in Grid Computing. It is also worth noting that virtualization – in the past – had significant performance losses for some applications, which has been one of the primary disadvantages of using virtualization. However, over the past few years, processor manufacturers such as AMD and Intel have been introducing hardware support for virtualization, which is helping narrow the performance

gap between application performance on virtualized resources as it compares with that on traditional operating systems without virtualization.

Monitoring: Another challenge that virtualization brings to the Clouds is the potential difficulty with fine-control over the monitoring of resources. Although many Grids (such as TeraGrid) also enforce restrictions on what kind of sensors or long-running services a user can launch, Cloud monitoring is not as straightforward as in Grids because Grids in general have a different trust model in which users via their identity delegation can access and browse resources at different Grid sites, and Grid resources are not highly abstracted and virtualized as in Clouds; for example, the Ganglia [31] distributed (and hierarchical) monitoring system can monitor a federation of clusters and Grids and has seen wide adoption in the Grid community. In a Cloud, different levels of services can be offered to an end user the user is only exposed to a predefined API, and the lower level resources are opaque to the user (especially at the PaaS and SaaS level, although some providers may choose to expose monitoring information at these levels). The user does not have the liberty to deploy her own monitoring infrastructure, and the limited information returned to the user may not provide the necessary level of details for her to figure out what the resource status is. The same problems potentially exist for Cloud developers and administrators as the abstract/unified resources usually go through virtualization and some other level of encapsulation, and tracking the issues down the software/hardware stack might be more difficult. Essentially monitoring in Clouds requires a fine balance of business application monitoring, enterprise server management, virtual machine monitoring, and hardware maintenance, and will be a significant challenge for Cloud Computing as it sees wider adoption and deployments. On the other

hand, monitoring can be argued to be less important in the Clouds as users are interacting with a more abstract layer that is potentially more sophisticated; this abstract layer could respond to failures and quality of service (QoS) requirements automatically in a general-purpose way irrespective of application logic. In the near future, user-end monitoring might be a significant challenge for Clouds, but it will become less important as Clouds become more sophisticated and more self-maintained and self-healing.

1.2 Motivation towards building a decentralized multi virtual Grid System

The main aspect for both Grid and cloud computing (i.e. decentralized multi virtual grid System): 1) Cloud computing provides transparency.2) Grid computing provides coordinated resource sharing. The common aim of both paradigms is to achieve a decrease in the need for additional expensive hardware and increase in computing power and storage capacities. Building a decentralized system for both Grid and Cloud computing (i.e. building a decentralized multi virtual Grid system), it is required to implement both, local resource management within each virtual organization and global resource management among the grid.

1.3 Problem statement

Building a decentralized multi-virtual grid system, which fulfills the requirements of both Grid computing and cloud computing, faces a number of issues and challenges for resource management. I consider the following main issues and challenges in my research.

1.3.1 Main issues

It is very essential for both local and global resource management that:

1) How can we assign decentralized allocation of tasks to suitable nodes to achieve both local and global load balancing?

2) How we can handle both regular node and broker failures?

1.3.2 Main Challenges

1) Stability with Scalability*:* How the system can achieve dependable performance with various loads of services and broker failure?; How the systems can maintain throughput under failure with the bigger environment to achieve load balancing and avoid job starvation?

2) System Transparency: How system can achieve transparency in a multi-virtual organization system in which the complexity of the entire system must be transparent to regular participants?

1.4 Hypothesis

Building a decentralized system for both Grid and Cloud computing can achieve both local and global load balancing and handle both regular node and broker failures by implementing both local resource management within each virtual organization, and global resource management among the grid. Moreover, decentralized systems can solve problems based stability and scalability and can achieve system transparency: 1) by implementing decentralized allocation of the task to suitable nodes for both local global

resource management (i.e. Efficient job scheduling). 2) By implementing efficient failure handling algorithm. 3) By choosing appropriate topologies at broker overlay.

1.5 Research Objectives

Design, implement and simulate a decentralized computing infrastructure that maintains stability with scalability, together with achieving system transparency for resource management. Develop two algorithms for both local and global resource management in order to achieve local and global load balancing and handling of both regular and broker failures. Two main algorithms are: 1) Service allocation algorithm. 2) Failure handling, and resource information sending/exchange algorithm. To evaluate the performance of both algorithms with a different broker overlay topologies in the presence of broker failures.

1.6 Research Contributions

The main contributions to this proposed framework are: 1) I have designed and proposed architecture as an infrastructure to maintain stability with scalability. 2) The proposed model retains the system decentralization and increase the scalability. 3) I have addressed the issue, based on decentralized allocation of the task to suitable nodes to achieve local and global load balancing and handling of both regular node and broker failures. For this, I have described two main algorithms: resource information exchange and service allocation algorithms.

Moreover, my proposed framework provides a cost effective alternative to hierarchical structured P2P systems requiring costly merging because it is allowing

exploring the whole overlay without the need for hierarchical systems due to one broker periodical allocation.

1.7 Thesis organization

The remainder of the thesis is organized as follows; Chapter 2 provides background studies, related works and identified issues and problems. Chapter 3 describes a brief overview of the proposed architecture by discussing services and several components involved in architecture design. It also describes the resource information exchange mechanism, service allocation model and failure handling mechanism. Chapter 4 describes the simulation model and presents the performed experiments and discussion of the results. Chapter 5 concludes the thesis, and as a future work, we propose other collaboration aspects in a multi- virtual organization environment.

CHAPTER 2

Background studies, Related works and Identified issues and problems

2.1 Background Studies

We studied the OGSA vision [32] of a broadly applicable and adopted framework for distributed system integration, virtualization, and management that requires the definition of a core set of interfaces, behaviors, resource models, and bindings. This document, produced by the OGSA working group within the Open Grid Forum (OGF) [33], provides a first version of this OGSA definition. It focuses on the requirements and the scope of important capabilities required for supporting Grid systems and applications in both e-science and e-business. The capabilities described are Execution Management, Data, Resource Management, Security, Self-Management, and Information. The description of the capabilities is at a high-level and includes, to some extent, the interrelationships between the capabilities. Grid systems and applications aim to integrate, virtualize, and manage resources and services within distributed, heterogeneous, dynamic "virtual organizations" [Grid Anatomy] [3], [Grid Physiology] [1]. The realization of this goal requires the disintegration of the numerous barriers that normally separate different computing systems within and across organizations so that computers, application services, data, and other resources can be accessed as and when required, regardless of physical location.

Further we studied a service-oriented architecture, the OGSA [32] that addresses this need for standardization by defining a set of core capabilities, and behaviors that address key concerns in Grid systems. These concerns raise several questions, including:

How do I establish identity and negotiate authentication? How is policy expressed and negotiated? How do I discover services? How do I negotiate and monitor service level agreements? How do I manage the membership of, and communication within, virtual organizations? How do I organize service collections hierarchically so as to deliver reliable and scalable service semantics? How do I integrate data resources into computations? How do I monitor and manage collections of services?

Furthermore, we studied the above compared infrastructure services and assumptions that constrain the development of the OGSA design, in particular it was explaining how OGSA builds on, and is contributing to the development of the growing collection of technical specifications that form the emerging Web Services Architecture [34, 35].

We also studied the current state of any work known to be underway to define such extensions or definitions. In [32], information is provided to the community regarding the specification of the Open Grid Services Architecture (OGSA). It does not define any standards, or technical recommendations i.e. distribution is unlimited.

Finally, we studied that the Open Grid Forum (OGF) [33] has embraced the Open Grid Services Architecture as the industry blueprint for standards-based grid computing. "Open" refers to the process used to develop standards that achieve interoperability. "Grid" is concerned with the integration, virtualization, and management of services and resources in a distributed, heterogeneous environment. It is "service-oriented" because it delivers functionality as loosely coupled interacting services aligned with industry-accepted Web service standards. The "architecture" defines the components, their

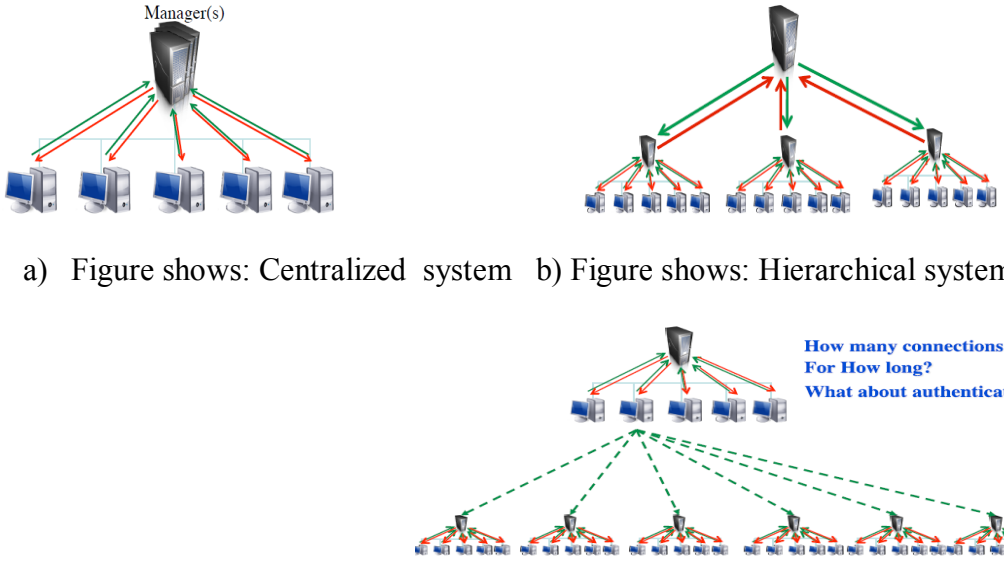
organizations, interactions, and the design philosophy used. OGSA-WG manages an architectural process of OGSATM 1 standards by working to collect requirements, evaluate the maturity of specifications, and produces periodic updates to OGSA informational documents and OGSA recommendation profiles.

The purpose for studying OGSA and others to understand their architectures. For example, OGSA does not denote a decentralized non-exclusive policy model and it provides information to the community but does not define any standards or technical recommendations.

2.2 Related Works

The resource management system is the central component of distributed network computing systems. There have been many frameworks [36, 37, 38, 39, 40, 41, 42, 51, 52, 53, 54, 55, 56, 57] that have focused on network computing, and have designed and implemented resource management systems with a variety of architectures and services. Basically, there are three types of distributed systems in most existing Grid systems: a) flat, b) Hierarchical and c) Interconnected. The three types of distributed systems are depicted in Figure 3.

In a flat organization, all distributed systems can directly communicate with each other without going through an intermediary. In a hierarchical organization, distributed systems at the same level can directly communicate with the distributed system directly above them or below them, or peer to them in the hierarchy. The fan out below a distributed system in the hierarchy is not relevant to the classification. Most current Grid systems use this organization since it has proven to be scalable. In an interconnected



a) Figure shows: Centralized system b) Figure shows: Hierarchical system

c) Figure shows: Interconnected system

Figure 3. Different types of distributed system.

structure, the distributed systems within the cell to communicate between themselves using flat organization.Designated distributed systems within the cell function act as boundary elements that are responsible for all communication outside the cell. The internal structure of a cell is not visible from another cell; only the boundary distributed systems are visible. Cells can be further be organized into flat or hierarchical structures. A Grid that has a flat cell structure has only one level of cells whereas a hierarchical cell structure can have cells that contain other cells. The major difference between a cell structure and hierarchical structure is as follows: a) an interconnected structure has a designated boundary with a hidden internal structure, b) whereas in a hierarchical structure, the structure of the hierarchy is visible to all elements in the Grid.

The resource model determines how applications and the resource management system (RMS) describe Grid resources. Distributed Systems, in most existing Grid systems are either flat [36, 37, 38, and 39] or hierarchical [40, 41, 42, 43, 44, 45, 46, and

35] in a single VO. Multi-VO model is implemented in some Grid systems: Arigatoni [40, 41, 42 44, 45 46, 47, 48] and interconnected systems [40, 42, 44, 45, 49, 50], and EGEE [51], and D4Science [52] implements centralized task allocation using a central broker. Grid3 [53], which is based on VOMS [54], implements centralized RM through management servers. DEISA [55] uses a central batch scheduler for task allocation. In [56], each VO implements the local RM model, and the framework implements centralized global RM. In NorduGrid [57], information about available resources is stored on dedicated database servers, and task allocation is carried out by local brokers on client nodes. None of these systems provide an efficient failure handling for both regular nodes and brokers. A detail description of some other most related resource management models is given below.

2.2.1 Condor: Cycle stealing technology for high throughput computing

Condor [36] is a high-throughput computing environment that can manage a large collection of diversely owned machines and networks. Although it is well known for harnessing idle computers, it can be configured to share resources. The Condor environment follows a layered architecture and supports sequential and parallel applications. The Condor system allocates the resources in the Condor pool as per the usage conditions defined by resource owners. Through its remote system call capabilities, Condor preserves the job's originating machine environment on the execution machine, even though if the originating and execution machines do not share a common file system and/or user ID scheme. Condor jobs with a single process are automatically check pointed and migrated between workstations as needed to ensure eventual completion. Condor can have multiple Condor pools and each pool follows a flat RMS organization.

The Condor collector, which provides the resource information store, listens for advertisements of resource availability. A Condor resource agent runs on each machine periodically advertising its services to the collector. Customer's agents advertise their requests for resources to the collector. The Condor matchmaker queries the collector for resource discovery which it uses to determine compatible resource requests and offers. Compatible agents contact each other directly and if they are satisfied the customer agents initiate computation on the resources. Resource requests and offers are described in the Condor classified advertisement (ClassAd) language [58]. ClassAds uses a semi-structured data model for resource description. The ClassAd language includes a query language as part of the data model, allowing advertising agents to specify their compatibility by including constraints in their resource offers and requests. Condor can be considered a computational Grid with a flat organization. It uses an extensible schema with a hybrid namespace. It has no QoS support, and the information store is a network directory that does not use X.500/LDAP technology. Resource discovery which is centralized queries with periodic push dissemination. The scheduler is centralized.

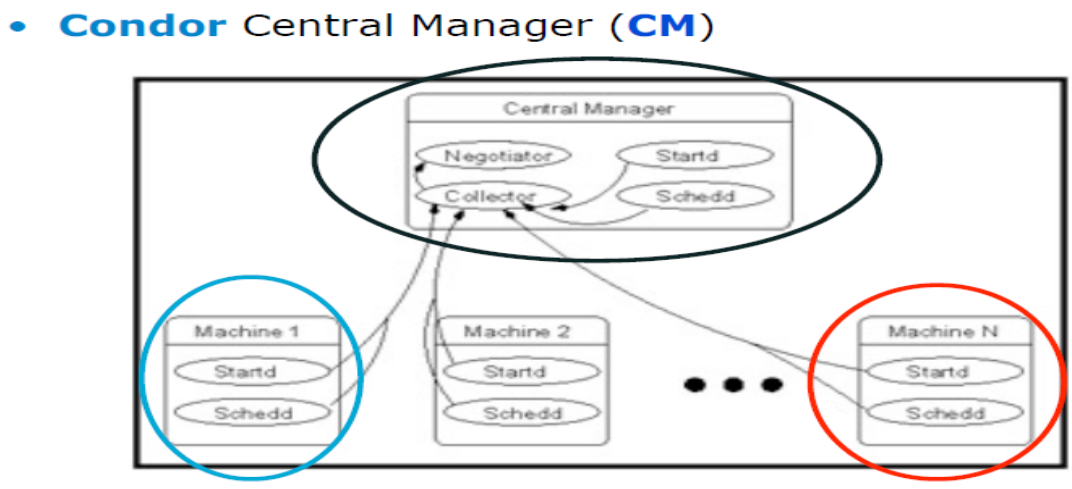


Figure 4. Condor work as a central manager [42].

2.2.2 Globus: A toolkit for Grid computing

The Globus system [59] enables modular deployment of Grid systems by providing the required basic services and capabilities in the Globus Metacomputing Toolkit (GMT). This toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, data management, resource reservation, and communications. Globus is constructed as a layered architecture in which higher level services can be developed using the low-level core services [60]. Its emphasis is on the hierarchical integration of Grid components and their services. Globus offers Grid information services via an LDAP-based network directory called Metacomputing Directory Services (MDS) [61]. MDS currently consists of two components: Grid Index Information Service (GIIS) and Grid Resource Information Service (GRIS). GRIS provides resource discovery services on a Globus based Grid. The directory information is provided by a Globus component running on a resource or other external information providers. The resource information providers use a push protocol to update GRIS periodically. GIIS provides a global view of the Grid resources and pulls information from multiple GRIS to combine into a single coherent view of the Grid. Globus is placed into the push resource dissemination category since the resource information is initially periodically pushed from the resource providers. Resource discovery is performed by querying MDS. Globus supports soft QoS via resource reservation [62]. The predefined Globus scheduling policies can be extended by using application level schedulers such as Nimrod/G, AppLeS, and Condor/G. The Globus scheduler in the absence of application level scheduler has a decentralized organization with an ad hoc extensible scheduling policy.

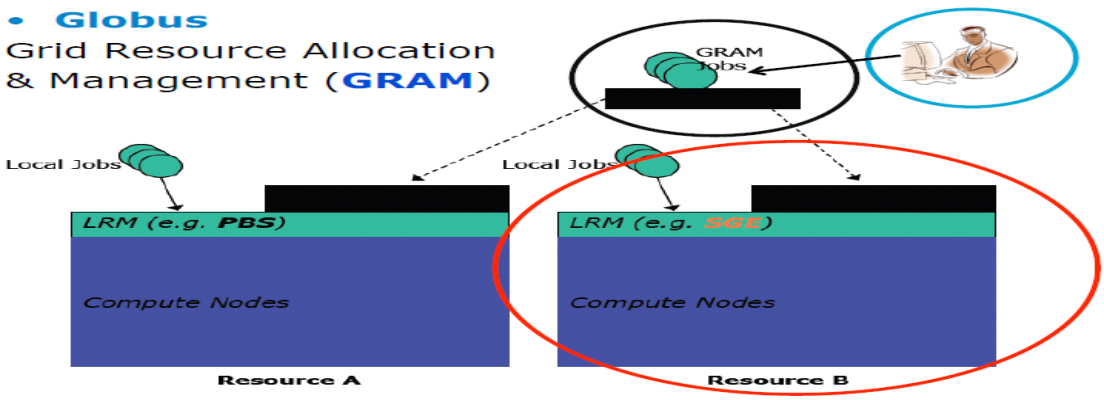


Figure 5. Globus- a toolkit for grid computing [59].

2.2.3 Nimrod/G: Resource broker and economy Grid

Nimrod/G [63, 64] is a Grid resource broker for managing and steering task farming applications such as parameter studies on computational Grids. It follows a computational market-based model for resource management. Nimrod/G provides support for the formulation of parameter studies, a single window to manage and control experiments, resource discovery, resource trading, and scheduling. The task farming engine of Nimrod/G coordinates resource management and results gathering. This engine can be used for creating user-defined scheduling policies. For example, Active Sheets are used to execute Microsoft Excel computations/cells on the Grid [65]. Nimrod/G is being used as a scheduling component in a new framework called Grid Architecture for Computational Economy (GRACE), which is based on using economic theories for a Grid resource management system. Nimrod/G has a hierarchical machine organization and uses a computational market model for resource management [66]. It uses the services of other systems such as Globus and Legion for resource discovery and dissemination. State estimation is performed through heuristics using historical pricing information. The scheduling policy is fixed-application oriented and is driven by user-

defined requirements such as deadline and budget limitations. Load balancing is performed through periodic rescheduling.

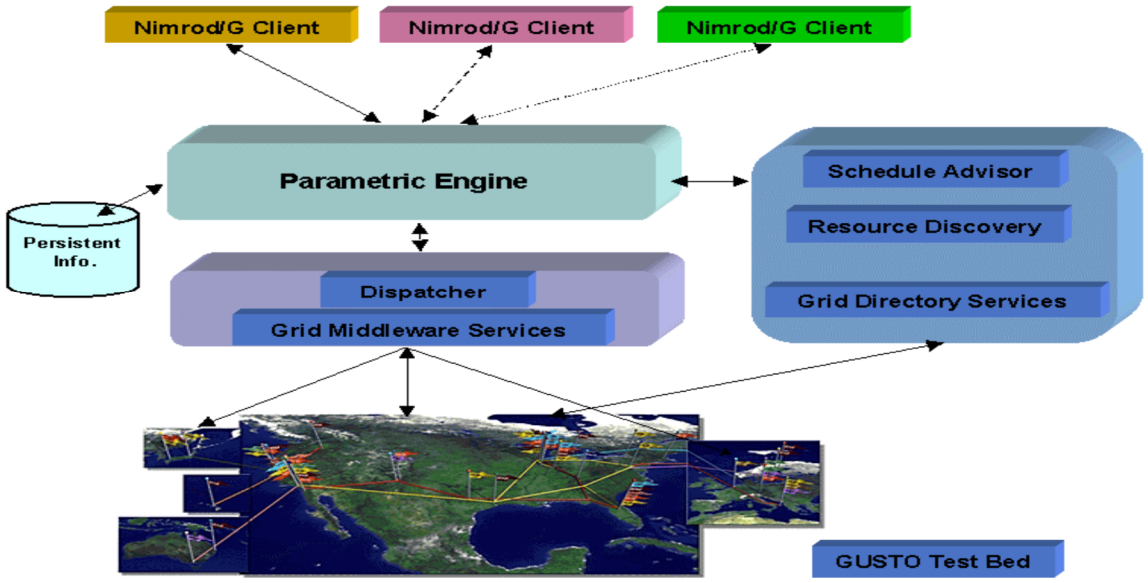


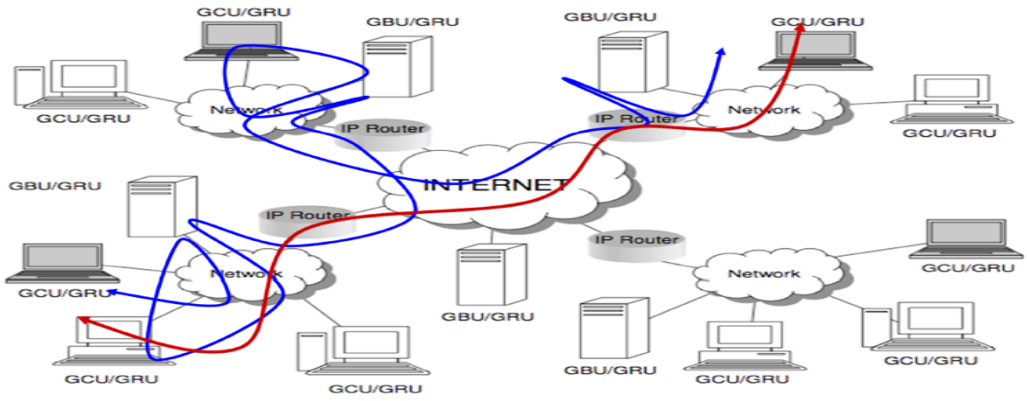
Figure 6. The Architecture of Nimrod/G [63].

2.2.4 Arigatoni Overlay Network: Super Broker and multi-virtual Grid system

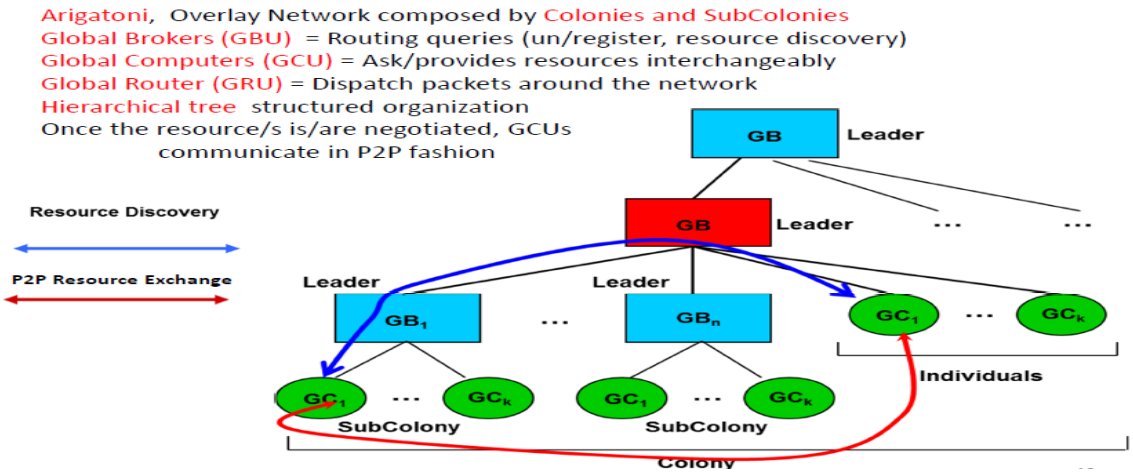
Arigatoni [40], a light weight model and a communication network called ArigatoNet that is suitable to deploy the Global Computing Paradigm over the Internet. I defined a simple but very efficient communication protocol, called Global Internet Protocol (GIP) on the top of TCP or UDP protocol [19]. Basic global computers and colonies of global computers can communicate by first registering to a brokering service and then by mutually asking for, or offering services. In this model, the resources are encapsulated in the colony in which they reside, and request for resources located in another colony traverse a broker-2-broker negotiation using a P2P overlay network. The model is suitable to fit with various global scenarios for classical P2P applications, like file sharing, band-sharing, memory space, to more sophisticated GRID applications like remote and distributed big (and small) computations, web services, computation

migration (i.e. ask to transfer one non completed the local run in another global computer unit(GCU) saving the partial results, under the case of a catastrophic scenario, like, e.g., fire, terrorist attack, earthquake etc., e.g. truly mobile ubiquitous computations) and Human computer interaction.

Compared to OGSA-based middleware [32] (e.g. Globus [33]), the Arigatoni model is much simpler and exploits the lower levels of the OSI stack. In principle, it could be deployed firstly in an intranet and further from intranet to intranet by overlapping an Overlay Network on the top of the actual network. For this, I could consider the Arigatoni model, with related middleware, as one prototypical example of overlay networks. Recall that an overlay network is an abstraction that can be implemented on top of a Global network to yield another global network. Overlay examples are resource discovery services (notion of resource sharing in distributed networks), search engines (abstraction of informal repository) or systems of trusted mobile agents (notion of autonomic, exploratory behavior) [67]. Since the Arigatoni model is P2P, it is worth noticing that a global computer unit (GCU) can also be a resource provider (or play both roles). Hence, a GCU can also be a supercomputer, a high performance visualizer (e.g. connect to a virtual reality center), or any particular resource provider that is linked to the Internet. This symmetry is another key feature of the Arigatoni model. Typically, a global computer unit (GCU) could ask for big computational power as when Grid users ask for memory space or for a particular piece of software. The Arigatoni model is depicted in Figure7.



a) ArigatoNet.



b) Arigatoni model based on Hierarchical tree.

Figure 7. Arigatoni Model [40].

Summarizing, the original contributions of the work are: a) A simple distributed communication model that is suitable to make resource discovery transparent; b) A Global Internet Protocol that allows Global Computers to negotiate resources; c) Complete independence with the classical scenarios of the arena, i.e. Grid, memory space, file/band sharing, web services, etc. This domain independence is a key feature of the model and for the protocol since it allows a complete abstraction from any given scenario.

2.2.5 Comparison between Arigatoni Model and OGSA Architecture

OGSA Architecture [32] proposes high level mechanisms or algorithms and does not address the overlaying Internet low level network protocols as I intend in my Arigatoni model [40]. Compared to OGSA-based architecture [32], the Arigatoni model is much simpler and exploits the lower levels of the OSI stack. In principle, at first it could be deployed in an intranet and then from intranet to intranet by overlapping an Overlay Network on the top of the actual network. Since the Arigatoni model is P2P, It is worth noticing that a global computer unit (GCU) can also be a resource provider (or plays both roles). Hence, a GCU can also be a supercomputer, and high performance visualizer (e.g. connected to a virtual reality center), or any particular resource provider which is linked to the Internet. This symmetry is another key feature of the Arigatoni model. Typically, a GCU could ask for big computational power, e.g. the Grid, ask for a particular piece of software or ask for memory space etc. However, OGSA architecture could not ask for big computational power. OGSA architecture [32] does not define a mechanism for devices to interoperate by offering services. However, Arigatoni model defines a mechanism for devices to interoperate by offering services. This way to understand common behavior of virtual organization has some theoretical basis on Game theory. Classical results from Game Theory are based on the assumption that basic shared currency connectivity (i.e. different resources such as CPU, Memory, Bandwidth, Data, etc.) is available and then the task is to design trustful mechanisms where users have an incentive to collaborate. Moreover, OGSA architecture [32] does not denote a decentralized non-exclusive policy model, and it provides information to the community but does not define any standards or technical recommendations. However, this means

that Arigatoni fits with motivations and cooperation behavior of different communities using ArigatoniNet. It tries to be policy neutral, leaving policy choices for each node at the implementation or configuration level, or at the community or organization level. Policy domains can overlap (one node can define itself as belonging "much" to colony foo and "a little bit" to colony bar). This denotes a decentralized non-exclusive policy model.

In OGSA architecture [32], extension does not define how to join a third party auction server. However, in Arigatoni, some Arigatoni extensions may define: 1) How to create and call third party services for on-line payment of services; 2) How to exchange digital cash for payment of services; 3) How to negotiate service condition between client and servant, including price and quality of service. The one-to-many natures of the SREQ GIP protocol requests are of particular interest in this case. An Arigatoni extension may define how to join a third party auction server. Candidate servant for a SREQ would contact the auction server and make their bid. The trusted auction server chooses the elected candidate and service conditions based on auction terms. The client would then contact the auction server and get this information. These extensions may take advantage of the GIP options' field, for example to transmit location and parameter information to call a third party system. Both Arigatoni model and OGSA architecture [32] can be extended to support various trust models. Moreover, reputation score could be expanded to a multiple-dimensional value, for example, adding a score for quality of the service offered by a node. However, Arigatoni encourages cooperation and enables gratuitous resource offering. But it may also suit for business extensions: a) A servant can sell resource usage, creating a resource business; b) A global broker unit (GBU) can sell

research service, creating a brokering business ("I point you to the best resources, more quickly than anyone else"). In Arigatoni model, the information about available resources is stored on dedicated database servers, and task allocation is carried out by local brokers on client nodes.

There are many different approaches and models for developing Grid resource management systems. The systems surveyed have for the most part focused on either a computational Grid or a service Grid. The other category of the system is the Grid scheduler such as Nimrod/G and AppLeS that is integrated with other Grid resource management system such as Globus or Legion. These combinations are then used to create application oriented computational Grids that provide certain levels of QoS.

2.3 Identified Issues and Problems

2.3.1 Identified Issues

Most existing Grid Systems fall within the following categories: a) Flat (Condor [36], Globus [41], gLite [68]; b) Hierarchical (Arigatoni [40], UNICORE [69], GridWay [70], BOINC [68]; c) Interconnected (Arigatoni [40, 41], Condor (flocking) [71], NorduGrid [57], EGEE [51]. All these architectures implemented centralized task allocation using a central broker; but, none of these systems provides an efficient failure handling for both regular nodes and brokers. For Examples, a) EGEE [51] implements centralized task allocation using a central broker while NorduGrid [57] the information about available resources is stored on dedicated database servers and task allocation is carried out by local brokers on client nodes.

Failure to service Issue in a centralized system: In centralized system, the entire network is controlled by a centralized broker. If the main broker fails, the entire network will undergo failure. If any resource fails the entire system will undergo partial failure. The centralized system is depicted in Figure 4.

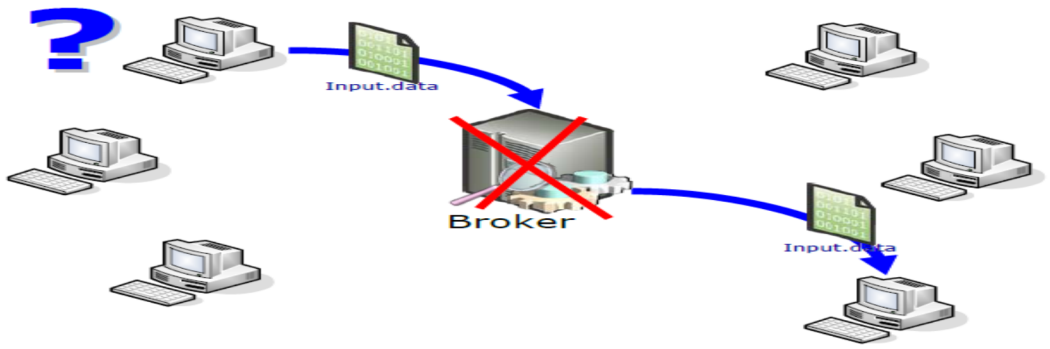


Figure 8. Failure services issues in centralized broker systems.

Designing issue: a) Typical issues in structure overlays are the size of each virtual organization (load balancing of the colony in the case of Arigatoni), and the internal coherence of the resources offered and requested by each colony (homogeneity of the colony). b) Typical bottlenecks of structure overlays are reliability, service availability (related to a few points of failure), and load balancing.

Routing, Scalability, transparency and security Issues: a) Crossing administrative barriers (Adm. Domains), Security (PKI certificate). b) Scaling up to the Large overlay computer, reliability (point of failure). c) Algorithms for routing requests and discover resources.

2.3.2 Problems

Effective use of computational grids via P2P systems requires up-to-date information about widely distributed resources. This is a challenging problem for very large distributed resources particularly taking into account the continuously changing state of the resources. Discovering dynamic resources must be scalable in number of resources and users and hence, as much as possible, fully decentralized. It should tolerate intermittent participation and dynamically changing status/availability. Many resource discovery algorithms and protocols have been proposed recently. As an example, in [72], the random forwarding algorithm has the advantage that no additional storage space is required for the node to record history. The learning-based algorithm performs constantly well. In Gnutella, the rather aggressive membership protocol maintains the highly dynamic nodes connected at a significant communication cost. Membership protocols based on epidemic communication mechanism are scalable with the number of participants. For example in [72], a P2Papproach to resource discovery in a grid environment is proposed. More precisely, the author presents a framework that guides the design of any resource discovery architecture. In [73], non-uniform information dissemination protocols are used to efficiently propagate information to distributed repositories, without requiring flooding or centralized approaches. Results indicate a significant reduction in the overhead compared to uniform dissemination to all repositories. In [74], a distributed resource discovery in the grid is proposed using a P2P network to distribute and query to the resource catalog. Each peer can provide resource descriptions and background knowledge, and each peer can query the network for existing resources. However, all these works propose high level mechanisms or

algorithms and do not address the overlaying Internet low level network protocols as intended in this work.

2.3.2.1 Problem in OGSA Architecture

OGSA architecture is not lightweight and provides complexity. Moreover it does not exploit the lower levels of the OSI stack. Considering these issues I have designed a model which is lightweight, much simpler and able to exploit the lower levels of the OSI stack. In OGSA architecture, there is a problem to define the mechanism for devices to interoperate, by offering services, in a cooperative manner based on reciprocity. Moreover, OGSA architecture does not denote a decentralized non-exclusive policy model and it provides information to the community, but it does not define any standards or technical recommendations.

I studied the OGSA architecture and found that it is not flexible enough to serve a mixture of different social structures, including: a) Independent end-user connecting through his/her ISP or migrating from hot-spot to hot-spot; b) Cooperating communities of disseminated people; c) More regulated or hierarchical communities (may be a better picture of the corporate network); d) Cooperative or competitive resource providers. In OGSA architecture, the extension did not define how to join a third party auction server.

2.3.2.2 Problem in Merging overlay Network

Some other work attempt to merge several overlay networks into one overlay network. The authors in [75] provide an analysis of the problem of merging two different overlays The authors in [76] introduce an algorithm of merging two rings based overlays however,

merging overlay require modifying the key space, as well as rearranging keys and data these tasks are expensive in terms of time and massages.

Summary of Identified Issues and Problems are shown below in Table1.

All these architectures implemented centralized task allocation using a central broker; but, none of these systems provides an efficient failure handling for both regular nodes and brokers. For Examples, a) EGEE [51] implements centralized task allocation using a central broker while NorduGrid [57] the information about available resources is stored on dedicated database servers and task allocation is carried out by local brokers on client nodes. b) Arigatoni [40] implemented centralized tasks allocation using a centralized broker within each virtual organization. If the main broker fails, the entire network will undergo failure. If any resource fails the entire system will undergo partial failure.

	Arigatoni	Globus	Condor	BONIC	glite	UNICORE	NorduGrid
Distributed Systems	Hierarchical tree	Flat	Flat	Hierarchical	Flat	Hierarchical	Hierarchical
Scheduling	Centralized	Centralized	Centralized	Centralized	Centralized	Centralized	Centralized
Handling Resources Failure	No	Yes	Yes	Yes	Yes	Yes	Yes
Handling Broker Failure	No	No	No	No	No	No	No

Table 1. Summary of Identified Issues and Problems.

CHAPTER 3

Overview of the Proposed Architecture and Related mechanism

3.1 Proposed Architecture

The proposed architecture for this thesis is based on global resource sharing within a collaboration of virtual organizations. Each virtual organization is set up as a domain. Each domain consists of one domain controller (i.e. Broker), and a collection of regular nodes. Fig. 9 shows the architecture of the Grid as collections of virtual organizations managed by a Broker Overlay structure.

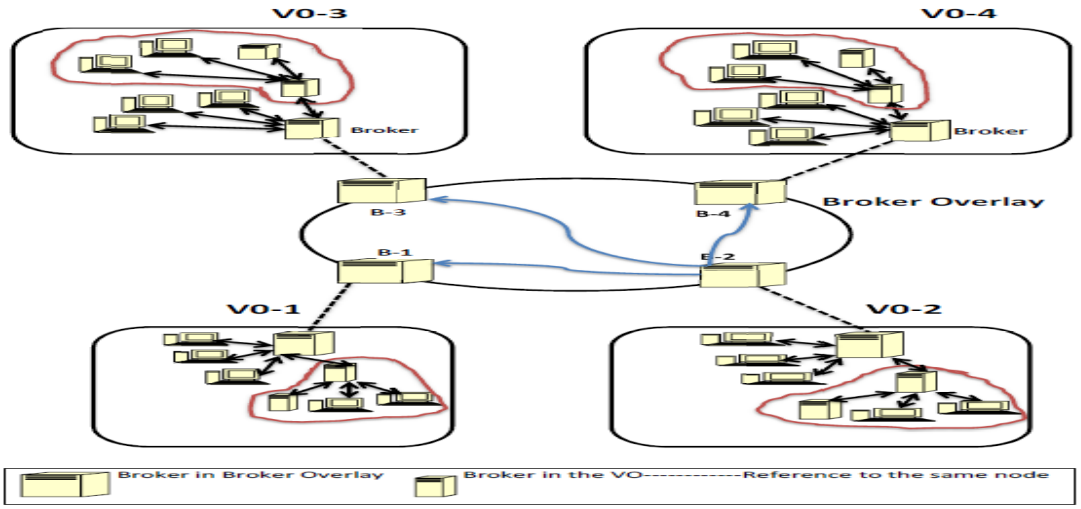


Figure 9. Proposed Grid architecture.

The proposed framework presents a decentralized multi-virtual resource management model based on hybrid peer-to-peer communication [40]. Rules of resource sharing within a virtual organization are well known by each node and controlled and managed by brokers. A broker is responsible for receiving requests for resources, comparing the requirements in each request with the resource specifications of the

available nodes, and direct requests to suitable nodes. Brokers from different virtual organizations construct a cooperative collection called, Broker Overlay. The idea is to provide each participating node with the ability to offer and claim computational resources. In addition, the complexity of the system is transparent to regular nodes in the broker overlay as each node interacts only with the attached broker. The regular node failures are managed using the same failure handling mechanism as Arigatoni does in [40]. In this thesis work, the broker failures are addressed.

3.1.1 Components Description

Components of the proposed framework are as follows:

A service in this architecture refers to a computational task. It has five execution parameters: 1) Required CPU, the computational power required for running the service. 2) Required Memory, the memory size required for running the service. 3) Expiration Time, the amount of time to wait before the allocation. 4) Creation Time, the time at which the service is created for the allocation. 5) Allocation attempts, the maximum number of attempts to deploy the service before it is expired.

A regular node refers to each non-broker node in the Grid. Each regular node can be a member of one virtual organization and can submit and/or run a service. A regular node is also responsible for periodically sending information about the current available resource state of the node to its broker. Each regular node has two resource parameters: 1) Available CPU, which refers to the available computational power in the node, and 2) Available Memory space. Regular is equivalent to Peer in Arigatoni[40], which contains

two components: Broker (Br), which is responsible for task execution, and Client (Cu), which is responsible for task submission1 [40].

A broker is a node which works as a virtual organization controller, can also work as a regular node in case of lack of available regular nodes. It is responsible for: 1) Allocating services to suitable nodes. A suitable node for a service is elected by performing a matchmaking process between the service requirements and the available resources of attached Grid nodes [24]. 2) Storing the current resource state for local nodes (i.e. in the same virtual organization) as well as global nodes (i.e. in other virtual organizations).

A virtual organization is an overlay of nodes which may be allocated in different regions and members of different organizations. Each VO is composed of one broker and regular nodes. Each VO is structured as a star logical topology, so that; communication is between the broker and regular nodes. There is no communication between regular nodes within the same virtual organization.

The broker overlay is the overlay network between brokers through which communication and data exchange between different virtual organizations is performed. For the broker overlay, four different network topologies are assumed: Ring, hypercube, star and fully connected. Based on the communication topology, each broker will have a number of neighbor brokers, those brokers with which direct communication can be established.

A Colony is a simple virtual organization composed by exactly one leader and some individuals. Individuals are regular node, or (sub) colonies. A simple definition of

the colony is given using the BNF syntax. Let Br denote the broker (the leader of the colony), and Cu denotes the regular node

Colony :: = { Br} / colony U {Cu} / colony U {colony}

The rules are: 1) every colony has exactly one leader Br and at least one individual (the Br itself); 2) every colony contains individuals (some Cu's, other colonies)

Some examples of the colonies are shown in Figure 10.

{Br} is a (small) colony

{Br1, Cu1…… Cum} is a colony

{Br1,Cu1….Cum, {Br2,Cum+1…….Cum+n}} is a colony (it contains subcolony).

{Br1,Cu1…Cum,Br2,Cum+1……Cum+n} is not a colony(two Br's)

{Br3, {Br1, Cu1…Cum}, {Br2, Cum+1..Cum+n}} is a colony (with two sub colonies)

{Br1,{Br1,Cu1..Cum},{Br2,Cum+1..Cum+n}} is a colony(Br1 is elected as the common leader)

{Br1,Cu1..Cum},{Br2,Cum+1..Cum+n}} is not a colony (no leader in the top level colony.

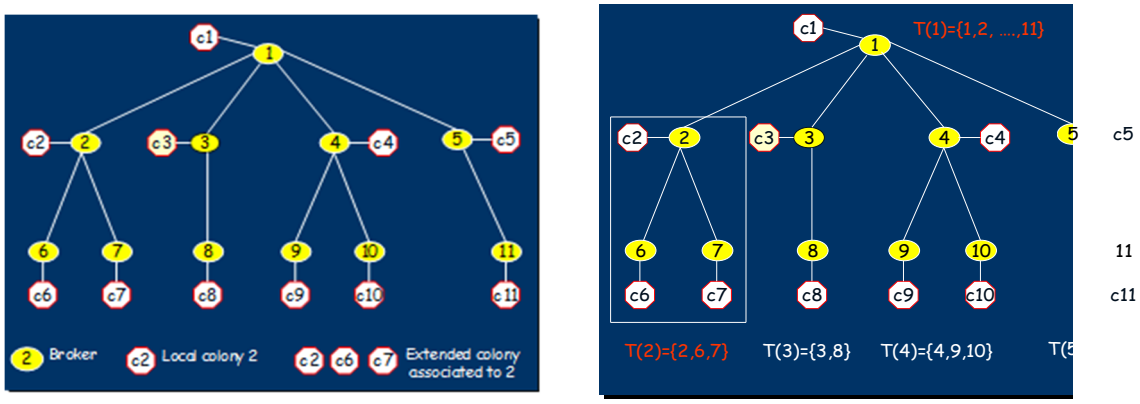


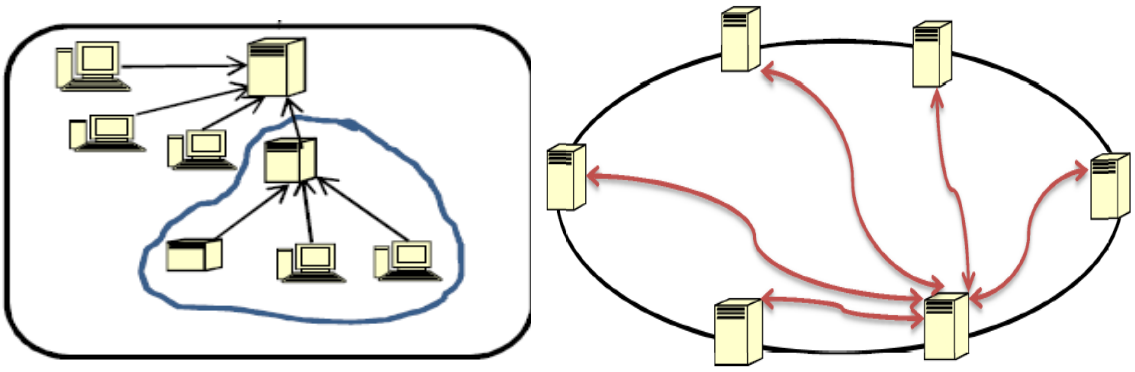
Figure 10. Some Colony's Examples.

3.2 Resource Information exchange mechanism

The broker overlay is the overlay network between broker through which communication and data exchange between different virtual organization is performed. The performance of the proposed framework depends on broker overlay topologies. Detailed mechanism has been described below in sub-section (3.2.1).

3.2.1 Resource Information exchange between broker to broker in broker overlay and between nodes and broker (i.e. within organization)

Resource information for each participating node is stored in a three field Resource Information Data Block, RIDB. The three fields represent: 1) Available CPU, 2) Available Memory, and 3) Time of last read. The third field, time of last read, is included to indicate if this read is too old so that it may not be dependable for allocation actions. Each broker maintains a set of RIDBs for all nodes in the system. Periodically, each regular node in a virtual organization reads the local current resource state (i.e. available CPU, and available Memory) in a data block and sends this block along with the reading time to its broker. Each time a broker receives a resource information block from a local node; it removes the previously stored reading, and replaces it with the current. Brokers also periodically exchange resource information through the broker overlay. Each broker performs one exchange operation with a single neighbor broker2 each time unit. The exchange operation is done by updating each resource information data set in each of the two brokers with the newest data blocks. The resource information model depicted in Figures11, and its algorithm is depicted in high-level form in Figure 12.



a) Resource information exchange within VOs. b) Resource information exchange between broker to broker

Figure 11. The resource information model.

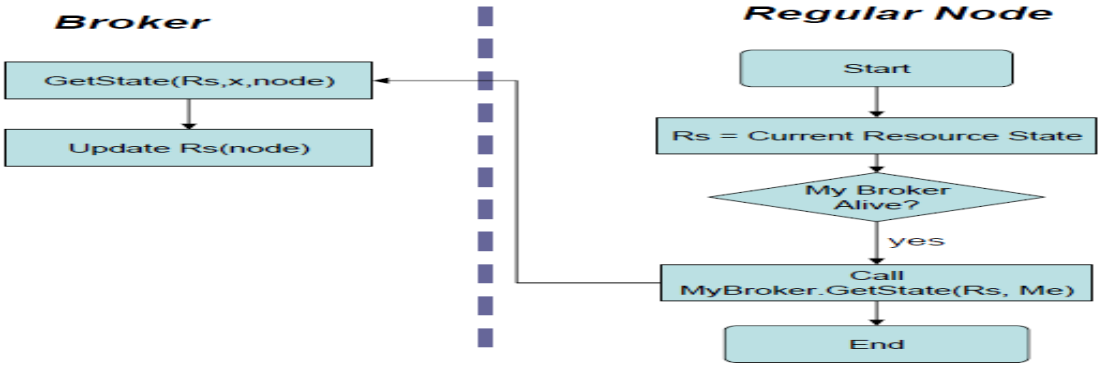


Figure 12. Resource information exchange algorithm.

Following is the stepwise explanation of resource information algorithm:

Sept 0: Regular node read the local current resource state in a data block.

Sept 1: Regular node sends the current available resource information to brokers in order to check whether my broker is alive or not.

Step 2: If the broker is alive, broker receives a resource information block from a local node.

Step 3 : Boker updates its data block with the new information.

3.3 Service Allocation model and Mechanism

Allocation of services to nodes is done through brokers. Submitting new services to brokers for allocation can be implemented in two ways: centrally, through a service allocation server connected to all brokers, or through the brokers by including a service allocation portal in each broker. In this work, allocation through the brokers is implemented. A service allocator component is included in each regular node for forwarding services to the attached broker. The service allocation model is depicted in Figure 13.

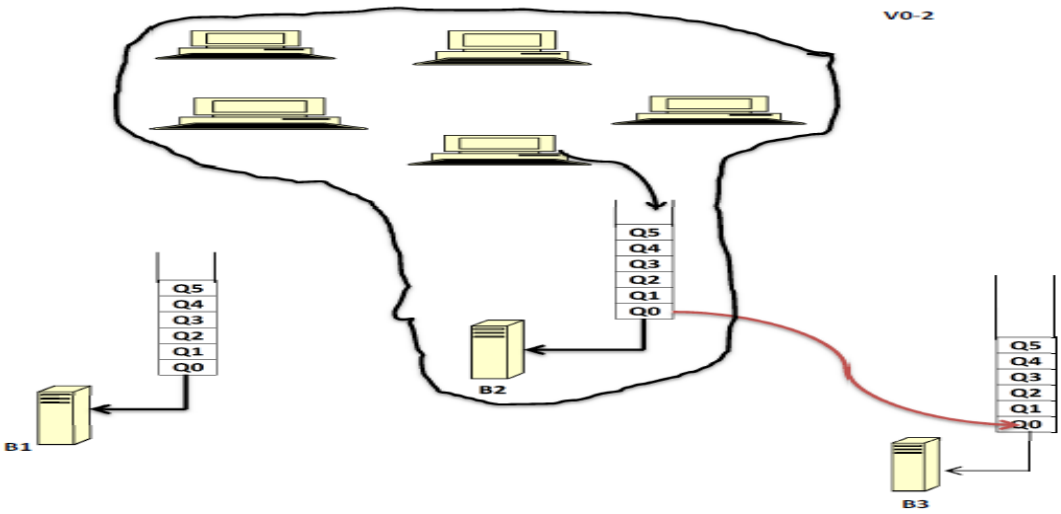


Figure 13. Service allocation Model.

Each broker has a service queue. When a service allocator sends a new service to a broker, it is automatically appended to the end of the service queue. Each time unit a broker picks the first service from the queue and starts a lookup process among the RIDBs, in order to find a suitable node with matching resource state of the resource requirements of the service. The Allocation algorithm is described in Fig. 14. The broker starts the lookup first among RIDBs of the local nodes. If no suitable resource found, the

broker repeats the operation among RIDBs of global nodes. If a global node matches, the broker passes the service to that node's broker with high priority so that it will be placed at the first position in the queue. The reason is to reduce the allocation time since there has been already previous allocation attempt(s). If there is no matching global node found, the service is passed to any neighbor broker, based on the topology. The allocation attempts parameter of a service is decremented each time the service is transferred to a new broker queue.

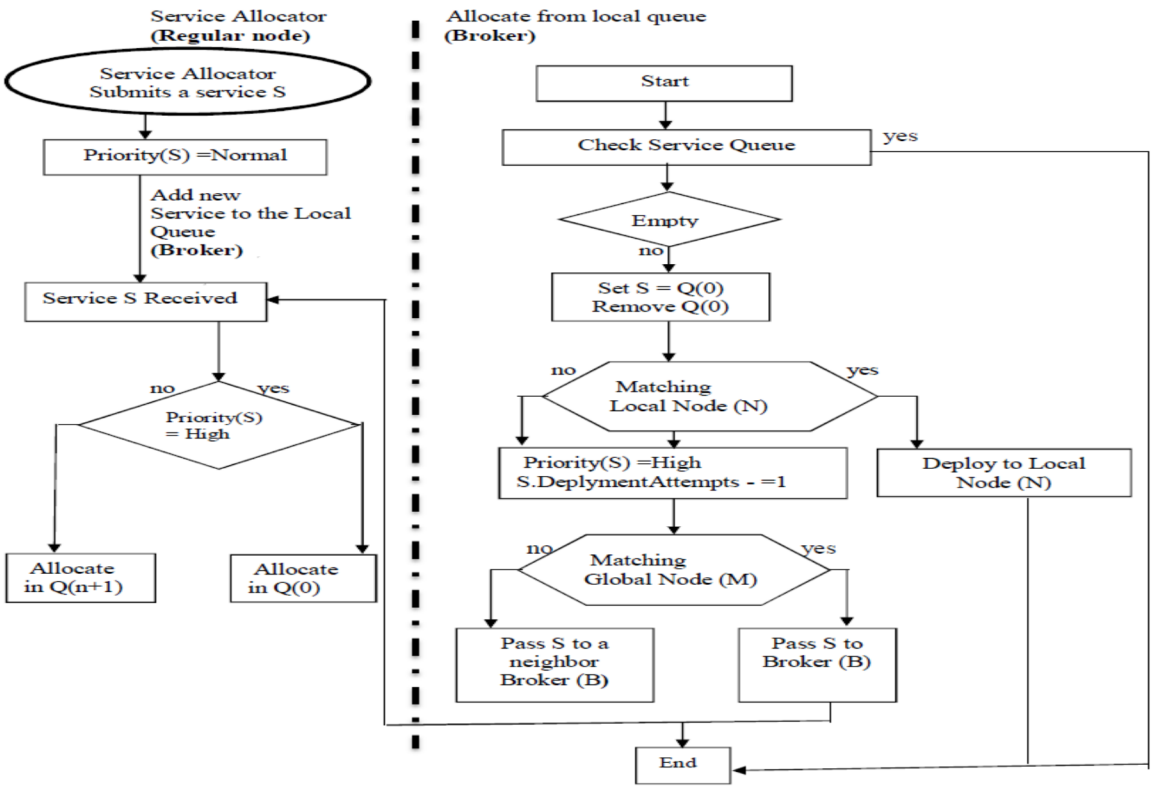


Figure 14. Grid Service allocation algorithm.

Following is the stepwise explanation of Service allocation algorithm:

Step 1: Each broker has service queue, first broker start checking whether its service queue is empty or not.

Step 2: If the service queue is empty, broker add new service to its local queue by requesting service allocator to submit a sevice S.

Step 3: If the service queue is not empty, broker picks the first service from the queue and start a lookup process among local resource information data block in order to find a suitable local nodes N with maching resource state to the resource requirement of the service.

Step 4: If suitable resource found, broker deploy to local node N.

Step 5: If no suitable resourse found, the broker repeats the operation among global node M by decrementing the allocation parameter 1(each time).

Step 6: If a global node M matches,the broker passes the service to that node's broker with high priority.

Step 7: If there is no matching global node found, the service is passed to any neighbor broker B.

Step 8: For service S, If the servicc expiration time less than current time minus service creation time, service S remove from service queue.

3.3.1 Service Validation Parameters

The Purpose of using expiration time and allocation attempts value to check maximum attempt before service get expire. The reason is to reduce the allocation time since there has been already previous allocation attempts. Detailed explanation already described in Section 3.3. Each time unit, a broker checks the expiration time and allocation attempts values for each service in the local service queue. For a service S:

If **(**S. ExpirationTime < (CurrentTime – S. CreationTime) OR

S. AllocationAttempts ==0**)**

// Service S is expired

Remove(S); //from the local service queue

3.4 Broker Failure Handling mechanism

Two types of failure are considered: regular node failure and broker failure. Regular node failures are managed in the same failure handling mechanism in Arigatoni [40, 48]. In this master thesis research, the focus is on the broker failure. In a virtual organization, it is assumed that each regular node has direct communication only with its broker. In addition, each node in the Grid holds a list of information about all existing brokers in the broker overlay. This information is updated periodically in regular nodes through their local brokers.

When a broker failure occurs, a regular node will detect the broker failure when it attempts to send its resource information to the broker. In case of broker failure, all regular nodes in the local virtual organization of the failed broker will be detached from the Grid. Once a broker failure is detected, a regular node sends a membership request to the first broker in the list. If the request is granted, the node will set the new broker as the attached broker, and adds it as a neighbor; otherwise the request is repeated to the next broker in the list. Fig 15a-15c shows failure handling steps and Fig. 15d describes the failure handling algorithm implemented in regular nodes. The algorithm is repeated each time unit.

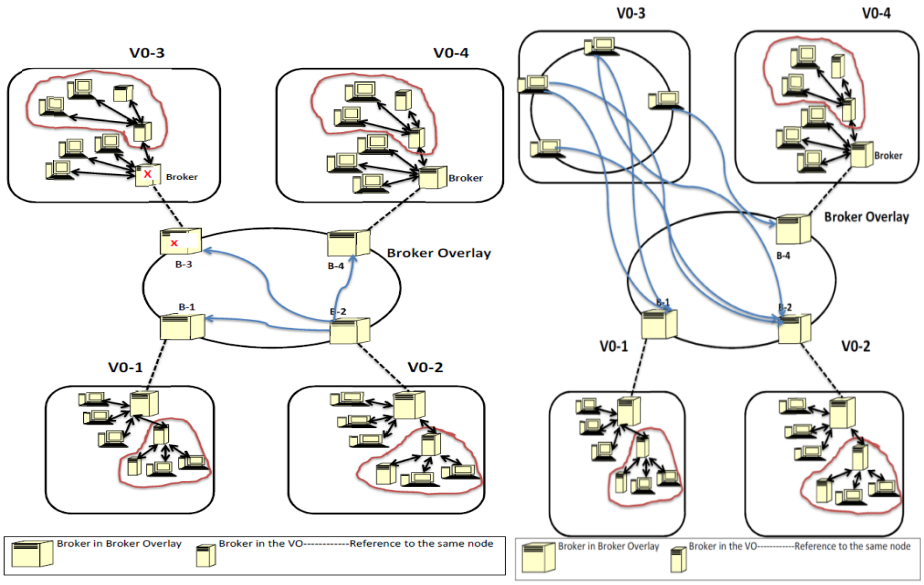


Figure 15. (a) Service broker failure occurs. Figure 15. (b) The detached grid node sends a membership request to the first broker in the list.

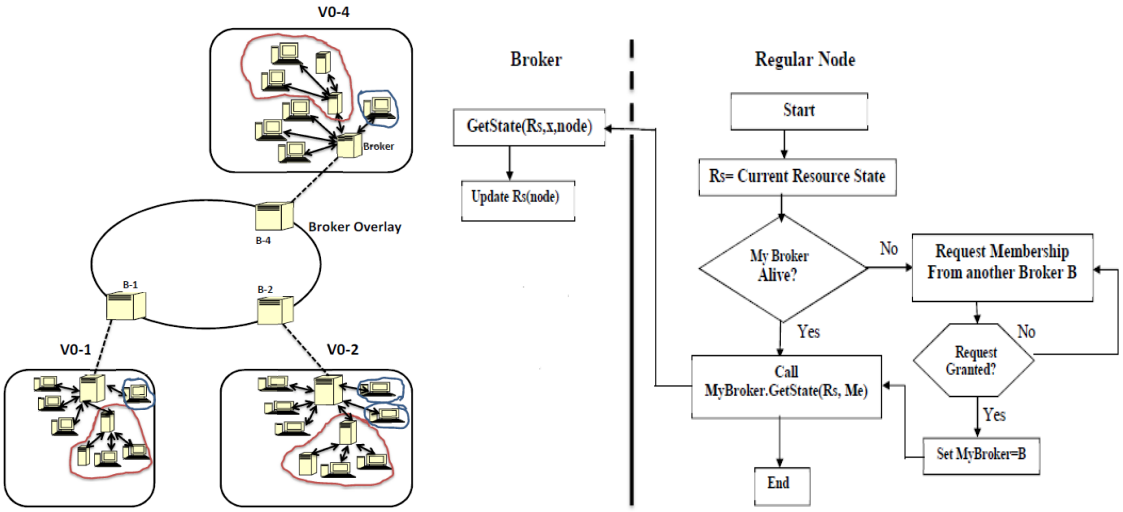


Figure15. (c) Regular nodes granted a membership request. Figure 15. (d) Failure handling algorithm and resource information sending algorithm.

Following is the stepwise explanation of Failure handling, and information sending algorithm:

Sept 0: Regular node reads the current available resource state in a data block (regular node data block)

Sept 1: Regular node sends the current available resource information to broker to check whether my broker is alive or not .

Step 2: If the broker is alive, broker receives a resource information block from a local regular node.

Step 3: Boker updates its data block with the new information.

Step 4: If the broker is not alive, each nodes detached from the grid and request membership from another broker B.

Step 5: If request granted, regular set broker B as leader Boker.

Step 6: If broker request not granted, it request membership from another boker in list until granted the request.

Step 7: Broker node updates its data block with new resource nformation.

CHAPTER 4

Simulation Model, Performance Evaluation, and Comparison of various research works with Proposed Framework

4.1 PeerSim

PeerSim [77], a Java-based simulation-engine designed to help protocol designers in simulating their P2P protocols, has been designed to support dynamicity and scalability, and it offers predefined models for P2P simulation. The engine consists of components which may be 'plugged in' and used a simple ASCII file based configuration mechanism which helps to reduce the overhead. This PerSim simulator supports two types of simulation, which includes cycle-based and event-based. The cycle based engine is simplified by ignoring transport layer in the protocol stack, and increase scalability. The event driven engine, which supports dynamicity, is more realistic but decreases scalability. It supports both structured and unstructured overlay network.

PeerSim simulation life-cycle: PeerSim was designed to encourage modular programming based on objects (building blocks). Every block is easily replaceable by another component implementing the same interface (i.e., the same functionality). Following are the objects, interface and cycle driven protocols classes.

a) Node object: The P2P network is composed of nodes. A node is a container of protocols. The node interface provides access to the protocols it holds, and to a fixed ID of the node.

b) CD Protocol: It is a specific protocol, that is designed to run in the cycle-driven model. Such a protocol simply defines an operation to be performed at each cycle.

c) Linkable: Typically implemented by protocols, this interface provides a service to other protocols to access a set of neighbor nodes. The instances of the same linkable protocol class over the nodes define an overlay network.

d) Control: Classes implementing this interface can be scheduled for execution at certain points during the simulation. These classes typically observe or modify the simulation.

4.2 Simulation Model

My proposed framework is based on structured overlay network. There are two types of communication and data exchange between the node and broker which include node broker, and broker to broker in an overlay network. Node to a broker and broker to broker communications are cycle based. PeerSim has been used in this model due to its support for cycle based simulation We also have built three different driven classes along with three reference classes for performance evaluation of the model. Under the simulation model, the proposed framework system can adapt at some extent to the service deploying load in order to achieve required performance.

4.2.1 Simulation Model Life-cycle

The life-cycle of a cycle-based simulation model is as follows. The first step is to read the configuration file, given as a command-line parameter. The configuration contains all the simulation parameters concerning all the objects involved in the experiment.

Then the simulator sets up the network initializing the nodes in the network, and the protocols in them. Each node has the same kinds of protocols; that is, instances of a protocols form an array in the network, with one instance in each node. The instances of

the nodes and the protocols are created by cloning. That is, only one instance is constructed using the constructor of the object, which serve as prototypes, and all the node cloned from this point.

At this point, initialization needs to be performed, that sets up the initial states of each protocol. The initialization phase is carried out by both GridAllocator and GridFailureControl classes that are scheduled to run only at the beginning of each experiment. In the configuration file, the initialization components are easily recognizable by the init prefix. The GridNode class are simply controls, configured to run in the initialization phase.

After initialization, the cycle driven engine calls all components (protocols and both GridAllocator, GridFailureControl) once in each cycle, until a given number of cycles, or until a component decides to end the simulation. Each classes in Simulation Model (GridAllocator, GridFailureControl and Protocols) is assigned a Scheduler class, which defines when they are executed exactly. By default, all classes are executed in each cycle. However, it is possible to configure a protocol or both GridAllocator and GridFailureControl to run only in certain cycles, and it is also possible to control the order of the running of the components within each cycle. When both GridAllocator and GridFailureControl collect data, they are formatted and sent to the standard output that can be easily redirected to a file to be collected for further work. Following is the detailed explanation for both reference and cycle driven protocol classes.

4.2.1.1 GridNode class: A reference for node objects. All the GridNode class created during the simulation are instances of classes that implement one or more interfaces.

4.2.1.2 GridAllocator and GridFailureControl classes: Both are included as references for Control objects, which simulate service allocation and failure handling.

4.2.1.3 Grid CD Protocol: It includes in each regular node, which is responsible for communicating with the attached broker and sending the resource information in each simulation cycle.

4.2.1.4 Allocation Protocol: It includes in each regular node, which is responsible for responding to the allocation requests from the broker.

4.2.1.5 Grid Broker Protocol: It is included in each broker node for performing the tasks associated with the broker.

4.2.1.6 The IdleProtocol: It is in the main PeerSim package, which is included in each node to be responsible for establishing communication with neighboring nodes. Fig. 16 describes the Grid simulation model and communication between different protocols.

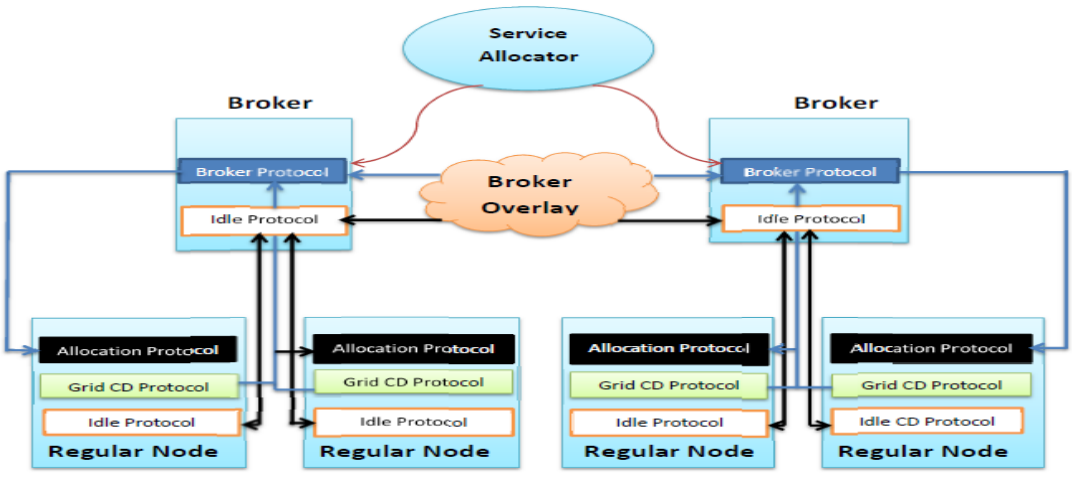


Figure 16. Grid simulation model

4.2.2 The purpose of using Simulation Model

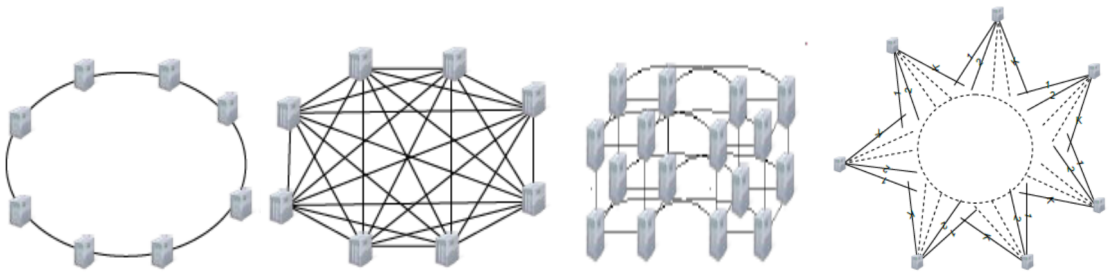
The main purpose of designing this simulation model is to evaluate the performance of the both algorithms with different overlay topologies. Since proposed framework is based on a decentralized multi-virtual resource management that can achieve both local and global load balancing. It also the handles both regular and broker failure by implementing an efficient failure handling and resource information exchange algorithms. Furthermore, it can solve problems based on stability, and achieves system transparency by implementing efficient service allocation algorithms.

Since a major cause of failure service is based on load balancing due to lack of data updation between broker to broker communications at the overlay network in the proposed framework, therefore, we have implemented above algorithms in order to solve this issue.

4.3 Performance factors

To evaluate the performance of the proposed architecture, three performance factors are used: Validity of stored resource information, Efficiency of service allocation, and Impact of broker failure on resource information updating. There are four topologies for the broker overlay have been used as shown in Figure 5: ring, star, fully connected (pure peer-to-peer) and hyper-cube.

Let **N** denote the total Grid size, and **M** are the number of virtual organizations. We performed experiments based on theses three factors in order to evaluate the performance of both algorithms with different broker overlay topologies.

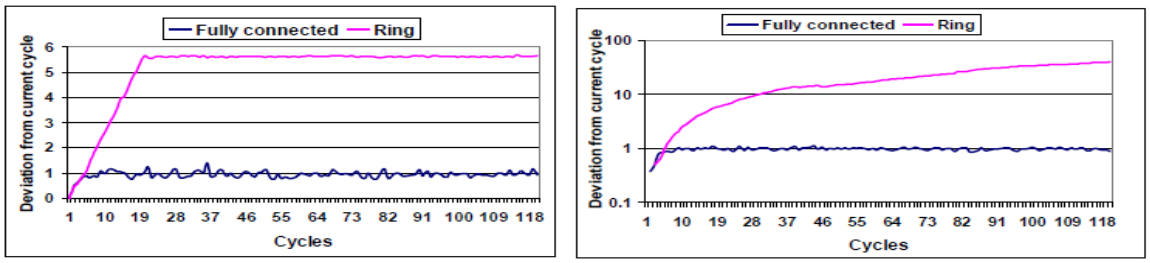


a) Ring b) Fully connected c) Hyper-cube d) Star

Figure 17. Shows four topologies for the broker overlay has been used for experimental results.

4.3.1 Validity of the stored resource information

The validity of the stored resource information is implemented through measuring the efficiency of the resource information exchange algorithm in keeping resource information up to date. The implemented methodology is to depict the deviation of the reading time values of RIDBs stored in the resource information data set, from the current cycle in a broker, with the simulation cycles. The results are read from one broker. For this performance metric, topologies for the broker overlay are ring and fully connected. A total of 120 simulation cycles has been used. Two experiments are performed with the following configuration: 1) N = 100, M = 20. 2) N = 500, M = 100. The results are shown in Figure 18.



Experimen1 N= 100, M=20 (small scale) Experiment 2 N=500, M= 100 (long scale)

Figure 18. Shows deviation of resource information reading time from the current cycle among simulation cycles.

4.3.1.1 Experimental results

As expected, figure 18 shows that the deviation is much more less for the fully connected topology than for the ring topology. In addition, when the network size and the number of brokers were increased, in experiment 2, the deviation remained at the same level for fully connected topology but increased for the ring topology. This can be attributed to the fact that, in a fully connected topology, all brokers are neighbors and can exchange resource information. This increases the probability of getting fresher data. In the ring topology, a broker has only two neighbors. Increasing the number of brokers, the number of broker neighbors increases for the fully connected topology but remains two for the ring topology. This reduces the chance of reaching data stored in far brokers (i.e. with the large number of hops between) in a ring topology, so, the deviation increases.

4.3.2 Efficiency of Service Allocation

The factor based on service allocation, we measure the efficiency of the allocation algorithm for distributing services among available suitable nodes, using different broker overlays. The network size is fixed to 500 nodes, and 100 virtual organizations. The implemented methodology is to depict the total number of waiting services, in broker queues, and the number of expired services with the simulation cycle. The results are collected from all brokers.

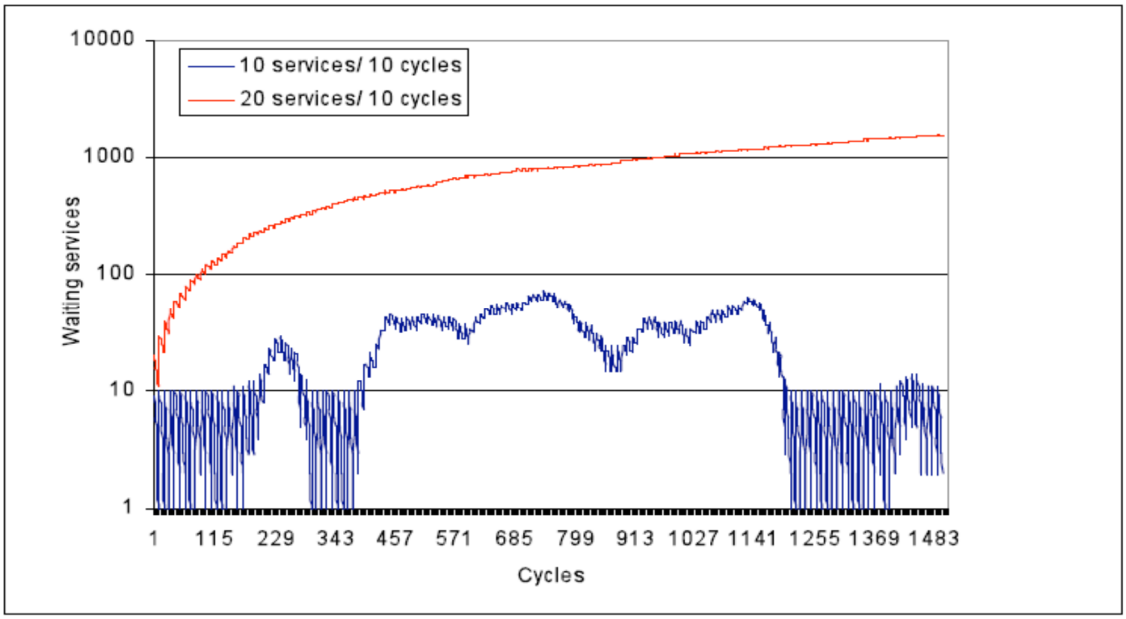


Figure 19. Number of waiting services plotted against simulation cycles for periodic allocation using a fully connected broker overlay topology, for 10 and 20 services per 10 cycles.

4.3.2.1 The main allocation method and results

The main allocation method is: One broker periodical allocation. In this method, nodes of one VO deploy a number of services to the broker each specific number of cycles. The idea is to focus all the allocation traffic on one broker as the worst case, to measure the efficiency of service exchange. Only the fully connected topology is tested with a total number of cycles of 1500. Two experiments are performed with the following configuration: 1) Total of 1500 services deployed as 10 services per 10 cycles. 2) A total of 3000 services deployed as 20 services per 10 cycles. The results are depicted for experiment 1 and experiment 2 in figure 19 using a logarithmic scale.

In Figure 19, it is clear that in case of allocating 10 services every 10th cycle, the system can produce a dependable performance. It is noticed that some bottlenecks can occur, but the system can recover. In case of allocating 20 services every 10th cycle, it is

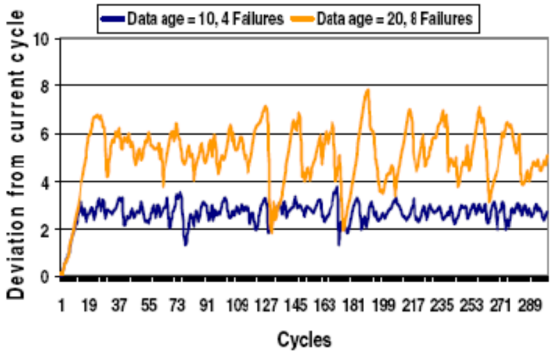
clear that the system becomes overloaded with service allocation requests. This occurs as a result of submitting all services to one broker. It can be concluded that, in periodical allocation, the allocation ratio of 10 services every 10th cycle (i.e. 1 Service/ cycle), is acceptable and can be handled in a Grid system of N >= 500, and 100 brokers with fully connected broker topology. If the ratio increased to 2 services/ cycle, the system, with the same network size would become overloaded.

4.3.3 Impact of Broker Failure on Resource Information Updating

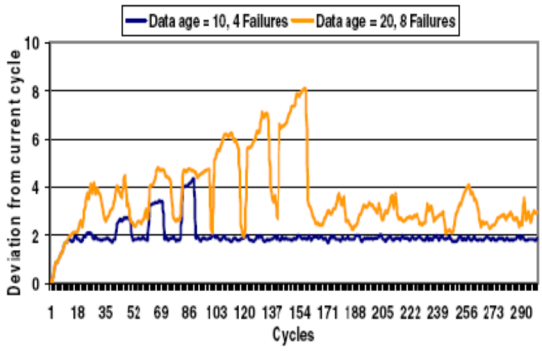
The aim of the experiments in this section is to measure the impact of broker failures on the validity of stored resource information. Experiment 2 in section 7.1 is repeated with adding injected broker failures during the simulation. With the existence of broker failures, it is expected that the deviation of the reading time values of RIDBs from the current cycle will increase due to failure. The reason is that resource information of the regular nodes which have been attached to the failed broker, will remain old and not updated until they are attached to other brokers and start sending resource information blocks. In the following experiments, a new parameter is taken into account: Data Age, the maximum age, in cycles, of resource information in a broker resource data set. In each simulation cycle, the broker protocol checks the reading time of each block in the resource information data set. If the reading time of a block is < (Current time – Data Age), then, this block is removed from the data set. If a new block for the same node is received later, in an exchange operation, it is added to the data set. The following experiments are performed by varying the value of Data Age.

Four topologies are used: ring, fully connected, star and hyper-cube. The network size is fixed to N = 500, and M = 100. The number of simulation cycles is 300. Two

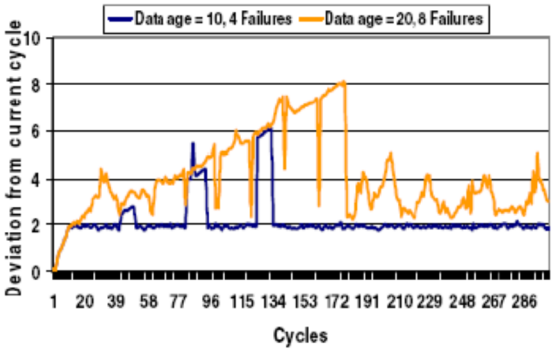
experiments are performed with varying the total number of failures: 1) Data age of 10 cycles with 4 injected broker failures, and 2) Data age of 20 cycles with 8 injected broker failures. The results are depicted in figure 20.



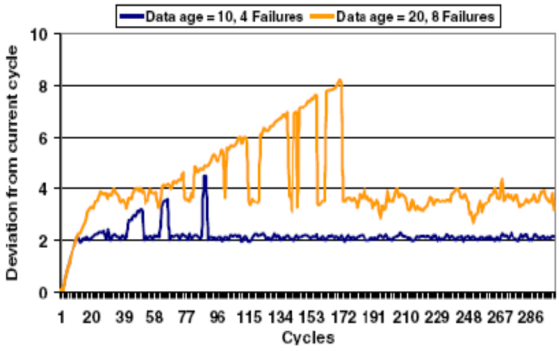
a) Ring broker overlay topology



b) Fully Connected broker overlay topology



c) Star broker overlay topology



d) Hyper-cube broker overlay topology

Figure 20. Shows impact of failures on the deviation of the resource information for data age of 10 cycles with 4 injected broker failures, and data age of 20 cycles with 8 injected broker failures.

4.3.3.1 Experimental Results

In Figure 20, it is clear that when the Data Age value decreases, the impact of failure decreases. This is because old data associated with unreachable nodes is periodically deleted from the resource information data sets. It is also clear that for fully connected, star and hyper-cube topologies, the system can recover from failures and return to a

stable state. In case of ring topology, the deviation has terrible variation and unstable. This can be described that, because of the lack of possible direct communications between brokers, it takes time for a broker to reach data stored in non-neighbor brokers.

It can also be noticed that the magnitude of deviation caused by failure increases each time a new failure occurs, in fully connected, star and hyper-cube topologies. This increase is not noticed in a ring topology. This increase can be described as follows: when a broker fails, all attached nodes attempt to join virtual organizations of other brokers. As the number of failures increase, the number of regular nodes attached to existing brokers also increase, so when a failure occurs then the number of detached nodes will be larger than those in the previous failures, which causes an increase in the number of old data blocks in brokers' data sets.

It can be concluded that the ring topology which is implemented in many hybrid peer-to-peer systems, is not applicable in case of assuming broker failures.

4.4 The Summary and Comparison of various research works

From the experimental results mentioned in Figure 19 and Figure 20, we observed that both algorithms performed well. The proposed framework retains the system decentralization and increases the scalability. The summary of the experimental results is as follows: a) Summary of the experimental results from the Service allocation algorithms with 1500 simulation cycles. The network size is fixed to 1500 nodes, and 100 virtual organizations.The results are shown in Table 2.

Total services deployed	Number of Services/Cycles	Results from Figure 19
1500	10/10	System performed well
3000	20/10	System becomes overloaded

Table 2. Summary of the experimental results 1.

b) Summary of the experimental results from failure handling algorithms with 300 simulation cycles. The network size is fixed to 500 nodes, and 100 virtual organizations.

Broker Topology	Ring	Fully connected	Star	Hyper-cube
Data Age of 10 cycles with 4 injected broker failure as shown in Figure	System can recover from failure	System performed well under broker failure	System performed well under broker failure	System can recover from failure
Data Age of 20 cycles with 8 injector broker failures as shown in Figure	System can't recover from failure	System can recover from failure	System can recover from failure	System can recover from failure

Table 3. Summary of the experimental results 2.

The summary and comparison of various research works with my proposed framework has shown in Table 4.

	My Frame work	Arigatoni	Globus	Condor	BONIC	glite	UNICORE	NorduGrid
Distributed Systems	Hierarchical	Hierarchical tree	Flat	Flat	Hierarchical	Hierarchical	Hierarchical	Hierarchical
Scheduling	Decentralized	Centralized	Centralized	Centralized	Centralized	Centralized	Centralized	Centralized
Handling Resources Failure	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Handling Broker Failure	Yes	No	No	No	No	No	No	No

Table4. Comparison of various research works with the Proposed framework.

I do believe that my approach from the experimental results, and simulation design is complementary to the proposed framework in the sense that it provides the basic infrastructure to real deployment of the broker overlay itself.

CHAPTER 5

Conclusions and Future Work

5.1 Conclusions

I have proposed architecture as an infrastructure to maintain stability with scalability. The proposed model retains the system decentralization and increases the scalability. A Grid simulation model, which is built based on the concept of collaboration of virtual organizations, has been presented. Global data exchange between virtual organizations has been implemented using the overlay network between brokers, based on different topologies. There are four topologies for the broker overlay has been discussed and implemented. Two main algorithms have been described: resource information exchange, and service allocation algorithm. I have compared various research works with the proposed work. We Performed experiments to evaluate the performance of both algorithms with different broker overlay topologies in the presence of broker failures. Results show that, the system can adapt to some extent to the service deploying load, and achieve required performance. The resource information exchange algorithm is efficient for the tested topologies, but in case of ring topology, it biases to instability in case of failures, and slow in updating resource information data due to the lack of possible direct communications between brokers.

The proposed model provides a cost effective alternative to hierarchical structured P2P systems requiring costly merging because it's allowing exploring the whole overlay without the need for hierarchical systems due to one broker periodical allocation used in

the experiment for checking the efficiency of service allocation and for resource information updating.

5.2 Future work

As part of future work, I have planned to propose an extended framework for resource discovery in Grid environments based on a hierarchical structured peer-to-peer architecture. The proposed organization will have the advantage of being scalable while providing fault-isolation, effective bandwidth utilization, and hierarchical access control. In addition, it will lead to a reliable, guaranteed sub-linear search which returns results within a bounded interval of time.

Another direction for future work, should focus on collaboration aspects within a multi-virtual organization environment encompassing security and rules of sharing, or policy.

REFERENCES

[1] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Available http://www.globus.org, 2002.

[2] K. Czajkowski, I. Foster, C. Kesselman, V. Sanger, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In Proceedings of the 8th Workshop on Job scheduling Strategies for Parallel Processing, July 2002.

[3] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15 (3), 2001.

[4] http://www.gridcomputingplanet.com/

[5] Bote-Lorenzo, M., Dimitriadis, Y. Gomez-Sanchez, E., Grid characteristics and uses: a grid definition. In Postproc. of the First European Across Grids Conference (ACG1/203), LNCS 2970, pages 291–298, Santiago de Compostela, Spain, Springer-Verlag, February 2004.

[6] Khanli, L. M., and Analoui, M. An approach to grid resource selection and fault management based on ECA rules. Future Generation Computer Systems, 24(4), 296-316, 2008.

[7] Yanbing Liu, and Yi Jian. Performance evaluation and modeling in grid resource management. Grid and Cooperative Computing, GCC 2006. Fifth International Conference, 335-338, 2006.

[8] Hao, T. A new resource management and scheduling model in grid computing based on a hybrid genetic algorithm. Computing, Communication, Control, and Management, 2008. CCCM '08. ISECS International Colloquium on, 3 113-117.

[9] Foster, I., Roy, A., and Sander, V. A quality of service architecture that combines resource reservation and application adaptation. Quality of Service, 2000. IWQOS. 2000 Eighth International Workshop on, 181-188.

[10] Yang, J., Bai, Y., and Qiu, Y. (2007). A decentralized resource allocation policy in minigrid. Future Generation Computer Systems, 23(3), 359-366.

[11] Aiken, B., Carpenter, B., Foster, I., Lynch, C., Mambretti, J., Moore, R., Strassner, J. Teitelbaum, B. (2000). Network Policy and Services: A Report of a Workshop on Middleware. Internet Engineering Task Force RFC 2768.

[12] Buyya, R., Venugopal, S. A gentle introduction to grid computing and technologies. CSI Communications9, 9–19, July 2005.

[13] Brock, M.; Goscinski, A., "Grids vs. Clouds," Future Information Technology (FutureTech), 2010 5th International Conference on, vol., no., pp.1, 6, 21-23 May 2010.

[14] Han Xingye; Li Xinming; Liu Yinpeng, "Research on Resource Management for Cloud Computing Based Information System," Computational and Information Sciences (ICCIS), 2010 International Conference on , vol., no., pp.491,494, 17-19 Dec. 2010.

[15] Haas, C.; Caton, S.; Chard, K.; Weinhardt, C., "Co-operative Infrastructures: An Economic Model for Providing Infrastructures for Social Cloud Computing," System Sciences (HICSS), 2013 46th Hawaii International Conference on, vol., no., pp.729, 738, 7-10 Jan. 2013.

[16] Sharma, B.; Thulasiram, R.K.; Thulasiraman, P.; Garg, S.K.; Buyya, R., "Pricing Cloud Compute Commodities: A Novel Financial Economic Model," Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on , vol., no., pp.451,457, 13-16 May 2012.

[17] Fu-yi group to talk about cloud computing pan development path, November 2008.

[18] http://docs.google.com/View?id=dgssm42h_13gm33jfdg.

[19] Islam, S.S.; Mollah, M.B.; Huq, M.I.; Aman Ullah, M., "Cloud computing for future generation of computing technology," Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2012 IEEE International Conference on , vol., no., pp.129,134, 27-31 May 2012.

[20] http://www.cnpaf.net/Class/g/200510/6544.html.

[21] Lei Wang; Jianfeng Zhan; Weisong Shi; Yi Liang, "In Cloud, Can Scientific Communities Benefit from the Economies of Scale?," Parallel and Distributed Systems, IEEE Transactions on , vol.23, no.2, pp.296,303, Feb. 2012.

[22] Cloud computing industry, top ten trends, bit Network Forum, http://server.chinabyte.com/278/8652778.shtml, December 23, 2009.

[23] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. “Falkon: a Fast and Light-weight task execution framework”, IEEE/ACM SuperComputing 2007.

[24] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large

Scientific Datasets", Jrnl. of Network and Computer Applications, 23:187-200, 2001.

[25] I. Foster, J. Vöckler, M. Wilde, Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation", SSDBM 2002: 37-46.

[26] A. Szalay, A. Bunn, J. Gray, I. Foster, I. Raicu. "The Importance of Data Locality in Distributed Computing Applications", NSF Workflow Workshop 2006.

[27] J. Dean, S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters", Sixth Symposium on Operating System Design and Implementation (OSDI04), 2004.

[28] I. Raicu, Y. Zhao, I. Foster, A. Szalay. "Accelerating Large scale Data Exploration through Data Diffusion", International Workshop on Data-Aware Distributed Computing 2008.

[29] K. Keahey, T. Freeman. "Contextualization: Providing One- Click Virtual Clusters", to appear, eScience 2008.

[30] K. Keahey, I. Foster, T. Freeman, and X. Zhang. "Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid", the Scientific Programming Journal, 2006.

[31] Ganglia, http://sourceforge.net/projects/ganglia/, 2008.

[32] H. Kishimoto and T.Maguire, "Open Grid Services Architecture Working Group," http://forge.gridforum.org/projects/ogsa-wg.

[33] Globus Alliance, "Globus Home Page," http:// www.globus.org/.

[34] Zhengqian Xu; Hongjun Dai; Feng Lu, "The Broker-Based Non-functional Support for Web Services on Embedded Systems," Advanced Software Engineering and Its Applications, 2008.

[35] AMrissa, M.; Ghedira, C.; Benslimane, D.; Maamar, Z.; Fayolle, J., "A mediation framework for Web services in a peer-to-peer environment," Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on , vol., no., pp.133,, 2005SEA 2008 , vol., no., pp.210,213, 13-15 Dec. 2008.

[36] Condor project, http://www.cs.wisc.edu/condor/.

[37] Luther, A., Buyya, R., Ranjan, R., Venugopal, S.: Peer-to-Peer Grid Computing and a.NET-based Alchemi Framework. Wiley Press, New Jersey (2005).

[38] Kooburat, T., Muangsin, V.: Centralized Grid Hosting System for Multiple Virtual Organizations, ANSCSE10 (March 22-24, 2006).

[39] Boloni, L., Jun, K., Palacz, K., Sion, R., Marinescu, D.C.: The Bond Agent System and Applications. In: Kotz, D., Mattern, F. (eds.) MA 2000, ASA/MA 2000, and ASA 2000.LNCS, vol. 1882, pp. 99–113. Springer, Heidelberg (2000).

[40] Amar Bahadur Patel, Arigatoni: Overlying Internet via Low Level Network Protocol, Master thesis KTH, Stockholm, Sweden, Annual report 2006. https: //www.kth.se/polopoly_fs/1.10674!annual_report.pdf.

[41] The Globus toolkit, http://www.globus.org/toolkit/.

[42] Open-source software for volunteer computing and grid computing, http://boinc.berkeley.edu/.

[43] Asia-Pacific Grid, http://www.apgrid.org/.

[44] D. Benza, M. Cosnard, L. Liquori, M. Vesin. Arigatoni: A Simple Programmable Overlay Network. In Proc. of JVA'06, John Vincent Atanasoff International Symposium on Modern Computing, Sofia, Bulgaria, pages 82--91, IEEE Computer Society, 2006.

[45] R. Chand, M. Cosnard, and L. Liquori. Resource Discovery in the Arigatoni Overlay Network. In I2CS, International Workshop on Innovative Internet Community Systems, LNCS. Springer Verlag, 2006.

[46] M. Cosnard, L. Liquori, R. Chand. Virtual Organizations in Arigatoni. In Proc. of DCM'06, 3rd International Workshop on Developments in Computational Models, Venice, Italy. Electronic Notes in Theoretical Computer Science, 22 pages, Volume 171, Issue 3, Pages 55-75, Elsevier, June 14, 2007.

[47] R. Chand, L. Liquori, and M. Cosnard. Improving Resource Discovery in the Arigatoni Overlay Network. In ARCS, International Conference on Architecture of Computing Systems, LNCS, pages 98–111. Springer Verlag, 2007.

[48] R. Chand, M. Cosnard, and L. Liquori. Powerful resource discovery for Arigatoni overlay network. Future Generation Computer Systems, 24(1):31–38, 2008.

[49] Vincenzo Ciancaglini, Rossano Gaeta, Riccardo Loti, Luigi Liquori:Interconnection of Large Scale Unstructured P2P Networks: Modeling and Analysis. ASMTA 2013: 183-197.

[50] Vincenzo Ciancaglini, Luigi Liquori, Giang Ngo Hoang, Petar Maksimovic: An Extension and Cooperation Mechanism for Heterogeneous Overlay Networks. Networking Workshops 2012: 10-18.

[51] EGEE: Enabling Grids for E-Science in Europe, http://public.eu-egee.org/.

[52] D4Science: Distributed collaborators Infrastructure on Grid Enabled Technology 4Science. http://www.d4science.eu/.

[53] Gardner, R.: Grid3: An Application Grid Laboratory for Science. In: Computing in High Energy Physics and Nuclear Physics 2004, Interlaken, Switzerland,September 27- October 1, p. 18 (2004).

[54] EU Data Grid Java Security Working Group. VOMS Architecture v1.1, http://grid-auth.infn.it/docs/VOMS-v1_1.pdf.

[55] Lederer, H., Pringle, G.J., Girou, D., Hermanns, M.-A., Erbacci, G.: DEISA: Extreme Computing in an Advanced Supercomputing Environment. NIC Series, vol. 38, pp. 687.

[56] Kooburat, T., Muangsin, V.: Centralized Grid Hosting System for Multiple Virtual Organizations, ANSCSE10 (March 22-24, 2006)688 (2007).

[57] NorduGrid: Nordic Testbed for Wide Area Computing and Data Handling, http://www.nordugrid.org/.

[58] Raman R, Livny M. Matchmaking: Distributed resource management for high throughput computing. Proceedings 7th IEEE International Symposium on High Performance Distributed Computing, July 1998.

[59] The Globus toolkit, http://www.globus.org/toolkit/.

[60] Czajkowski K, Foster I, Kesselman C, Karonis N, Martin S, Smith W, Tuecke S. A resource management architecture for meta computing systems. Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[61] Fitzgerald S, Foster I, Kesselman C, von Laszewski G, Smith W, Tuecke S. A directory service for configuring high performance distributed computations. Proceedings Sixth IEEE Symposium on High Performance Distributed Computing,1997; 365–375.

[62] Foster I, Roy A, Sander V. A quality of service architecture that combines resource reservation and application adaptation. Proceedings of the 8th International Workshop on Quality of Service (IWQoS '00), June 2000.

[63] Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid. Proceedings

of the International Conference on High Performance Computing in Asia–Pacific Region (HPC Asia 2000),2000.

[64] Kacsuk, P., Podhorszki, N., Kiss, T.: Scalable desktop Grid system. In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006, LNCS, vol. 4395, pp. 27–38. Springer, Heidelberg (2007).

[65] Kotler L, Abramson D, Roe P, Mather D. Active sheets: Super-computing with spreadsheets. Proceedings of the 2001 High.Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference, April 2001.

[66] Buyya R, Giddy J, Abramson D. An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications. Proceedings of the 2nd International Workshop on Active Middleware Services (AMS '00), August 2000.

[67] V. Sassone, "Global Computing II: A New FET Program for FP6," Talk,Bruxelles,4/6/04,http:www.cogs.susx.ac.uk/users/vs/research/paps/gc2InfoDayPres.pdf.

[68] Marzolla, M.; Andreetto, P.; Venturi, V.; Ferraro, A.; Memon, S.; Memon, S.; Twedell, B.; Riedel, M.; Mallmann, D.; Streit, A.; van de Berghe, S.; Li, V.; Snelling, D.; Stamou, K.; Shah, Z.A.; Hedman, F., "Open Standards-Based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE," e-Science and Grid Computing, IEEE International Conference on , vol., no., pp.592,601, 10-13 Dec. 2007.

[69] D. Anderson, Boinc: A system for public resource computing and storage. Proceedings of the 5th IEEE/ACM International GRID Workshop, pp. 1 7, 2004.

[70] Visegradi, A.; Acs, S.; Kovacs, J.; Terstyanszky, G., "Application repository based evaluation of the EDGI infrastructure," MIPRO, 2012 Proceedings of the 35th International Convention , vol., no., pp.295,300, 21-25 May 2012.

[71] Butt, A.R.; Rongmei Zhang; Hu, Y.C., "A Self-Organizing Flock of Condors," Supercomputing, 2003 ACM/IEEE Conference, vol., no., pp.42, 42, 15-21 Nov. 2003 doi: 10.1109/SC.2003.10031.

[72] A. Iamnitchi, I.T. Foster, and D.Nurmi, "A Peer-to-Peer Approach to Resources Location in Grid Environment," in proc. Of High Performance Distribute Computing, HPDC, 2002, p.419, full version. In http://www.cs.uchicago.edu/files/tr_authentic/TR-2002-06.pdf.

[73] V.Iyengar, S. Tilak, M.J.Lewis, and N.B. Abu-Ghazaleh, "Non-Uniform Information Dissemination for Dynamic Grid Resource Discovery," in Proc. of Network Computing and Applications, NCA. IEEE, 2004.

[74] F.Heine, M.Hovestadt, and O.Kao, "Towards Ontology-Driven P2P Grid Resource Discovery". In proc. of International Workshop on Grid Computing, GRID, IEEE/ACM, 2004, pp.76-83.

[75] A. Datta and K. Aberer, "The challenges of merging two similar structured overlays: A tale of two networks," in Proc. of EuroNGI '06.

[76] T. M. Shafaat, A. Ghodsi, and S. Haridi, "Dealing with network partitions in structured overlay networks," Peer-to-Peer Networking and Applications, vol. 2, no. 4, 2009.

[77] Montresor, A., Jelasity, M.: PeerSim: A Scalable P2P Simulator, http://peersim.sourceforge.net/.

VITA AUCTORIS

NAME:	Amar Bahadur Patel
PLACE OF BIRTH:	Allahabad, Uttar- Pradesh, India
YEAR OF BIRTH:	1976
EDUCATION:	C.S.J.M. University,B.Tech., Kanpur, India, 2001
	The Royal Institute of Technology (KTH), M.Sc. Stockholm, Sweden, 2006