Autonomic Resource Discovery service Infrastructure for Grid (ARDIG)

By

NC Mudassar Ali (Syndicate Leader)
PC Ammarah Kahlon
PC Shoaib Anwar

Submitted to the Faculty of Computer Science Military College of Signals
National University of Sciences and Technology, Rawalpindi In partial fulfillment for the requirements of a B.E. Degree in Computer Software Engineering

March 2008

Abstract

Traditional Grid monitoring and discovery services are mainly responsible to discover shared computing resources spanning multiple administrative domains and present their unified functionality to the users. Often these discovery services are not fault tolerant, and popular Grid middleware deploy discovery services which do not tolerate high resources churn. Moreover, some Grids deploy centralized discovery service solutions which severely limit the scalability of the Grid. Although solutions such as heart beat systems for dynamic resource availability, super peer discovery services which cater for scalability, have been devised to address this problem. However no unified discovery architecture has been proposed and implemented for next generation dynamic, decentralized, pervasive Grids. In this report, architecture is proposed which integrates an autonomic infrastructure into Grid monitoring and discovery services in order to provide for a more flexible, robust and self-healing Grid. The proposed architecture will lead towards a generic implementation which will be used to transparently extend existing discovery services.

Declaration

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institute or elsewhere.

Acknowledgements

Above all we must thank to Allah Almighty for giving us power, ability and opportunity to complete this challenging task. And after this we are grateful to our parents who provided us with their full support and spiritual guidance not only during this project but throughout our course work at the Military College of Signals. Also we are highly grateful to our internal supervisor, **Athar Mohsin Zaidi**, for having confidence in us and providing us initial impetus of taking up this task and for extending his outmost technical support and guidance throughout our project. We also express special thanks to **Dr Ashiq Anjum (CERN)** who supervised and managed all the research activities related to the project, and we must say thanks to Mr. Irfan Habib for his devoted guidance and support.

Achievement

“Autonomic Grid Discovery Service Infrastructure”

Paper accepted in the 3th *IEEE International Conference on Grid and Cooperative Computing*, May 2008, Kunming, China

Table of Contents

List of Figures

List of Graphs

Chapter 1

1. Introduction

1.2. Introduction

The heterogeneity and growing nature of Grid has made it difficult for Grid System to have optimal working environment at all time. Nodes membership is dynamic, leading to suboptimal organization of Grids, network congestion and issue of load balancing which effects Node performance and availability.

The Grid solutions so far are aimed towards fulfilling Grid standards such as GGF standards in order to make it efficient but these approaches have some short comings. e.g., such a system lacks intelligence as most of the decisions from establishing and configuring to maintenance are done manually. For it to be more effective and adaptable such system approach should be focused towards autonomic Grid services hence user's participation in many complex configuration and maintenance decision requiring extensive knowledge about the Grid would be minimized.

This project chooses an approach which aims at improving the existing grid infrastructure by incorporating autonomic discovery infrastructure. The goal is to provide it with an adaptable discovery service in its environment. This discovery service would be able to not only discover the resources for any Grid application; it could also manage the Grid system which includes healing the network if any faults occur and organizing the Grid system in order to optimize the performance of Grid. This same thing is summarized in the figure1.1.

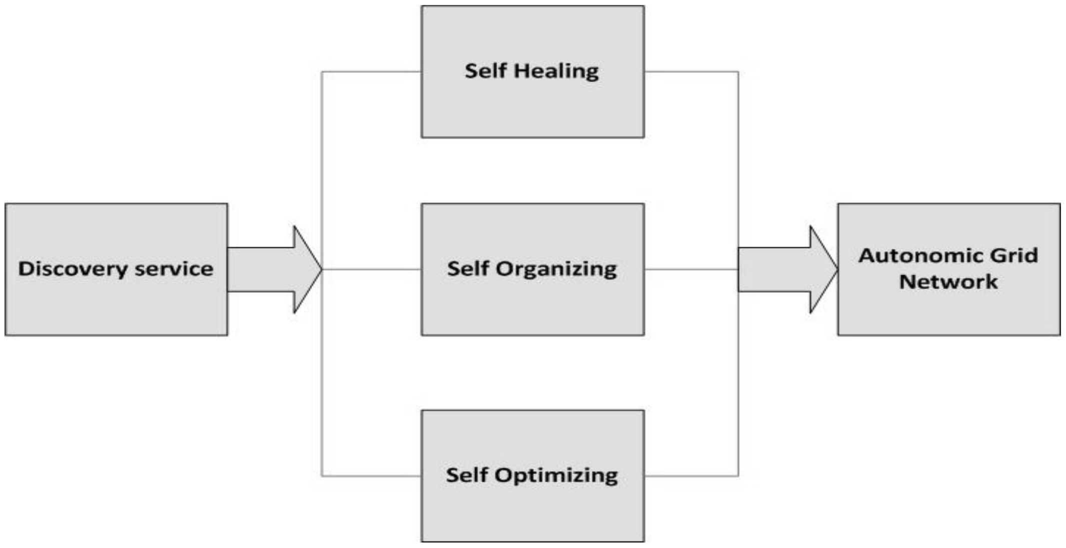


Figure 1.1: Objective of Autonomic Resource Discovery

1.3. Problem Statement

To develop a decentralized autonomic infrastructure and integrate it into Grid monitoring and discovery services in order to provide for more flexible, robust and self-healing Grids.

1.4. Scope

Building a unified architecture for decentralized self-optimizing, self-healing and self-organizing pervasive Grids, implement a decentralized resource broker and scheduling algorithm, distributed Heart Beat monitoring to cater for scalability and robust architecture and develop virtual organization creation system which would facilitate the scheduling and execution of Grid applications by aggregating Sub Grids into a single virtual Sub Grid, electing a leader which would manage the scheduling and eventual execution.

1.5. Related Work

In the current Grid climate in which there are Grid middleware, clusters software which provide such Grid facility like Condor and Grid Operation System in which, the level of autonomy is basic as they only make decision on some scenario which they are configure to do so. Among Grid middleware Globus [1], Glite are the most famous ones.

Current version of Globus which is Globus 4which uses a method known as MDS4 which is responsible for monitoring and discovery of resources. It uses indexing services as an aggregator service. This is basically a registry like UDDI which collects information and publishes them so the aggregator source (consumer) can use it, triggering services which collects and compares information to make decision and archiving services. The index service especially the trigger service provides some level of autonomy but these services are not tolerant to harsh and unforeseen environment.

Glite uses Service Discovery module which is not yet fully grown enough to be a separate service in Glite Architecture instead it is provided as client library. And it is coupled with Information and Monitoring Service so it cannot be modified without causing changes to the other modules. This Information Monitoring Service is based on producer and consumer approach so only that produced information matters which would caused significant amount of changes in order to incorporate new information or constraints.

Other autonomic projects on Grid are as follows .Optimal Grid Middleware which is a research prototype of grid-enabled middleware designed to hide complexities of partitioning, distributing, and load balancing. (IBM Almaden Research Center,San Jose,California), Organic Grid which is biologically inspired and fully-decentralized approach to the organization of computation that is based on the autonomous scheduling of strongly mobile agents on a peer-to-peer network. (Ohio State University) and AutoMate, which enable the development of conceptual models and implementation architecture to enable self managing Grid application autonomic (TASSL, a research lab at Rutgers University)

1.6. Organization of project report

Chapter 2, deals with the literature stuff of the project followed by Chapter 3 which is about the overall architecture of ARDIG and how components relate to each other and phantom OS and then comes Chapter 3 to 8 which starts with the series of chapters which methodically deal individual components developed in Project. Finally Chapter 9 is about the conclusion.

Chapter 2

2. Literature Review

2.2. Introduction

The chapter gives background knowledge of Grid Computing, its architecture and its standards as well about autonomic Computing, its feature and its composition. Detailed overview about the foundation concepts of autonomicy in Grids and its evolution in the research world is also given.

2.3. Grid Computing

Grid computing (or, more precisely a "grid computing system") is a virtualized distributed computing environment. Such an environment aims at enabling the dynamic "Runtime" selection, sharing, and aggregation of (geographically) distributed autonomous resources based on the availability, capability, performance, and cost of these computing resources, and simultaneously, also based on an organization's specific baseline and/or burst processing requirements. When people think of a grid, the idea of an interconnected system for the distribution of electricity, especially a network of high-tension cables and power stations, comes to mind. In the mid-1990s the grid metaphor was applied to computing, by extending and advancing the 1960s concept of "computer time sharing." The grid metaphor strongly illustrates the relation to, and the dependency on, a highly interconnected networking infrastructure.

To solve mainly scientific computing and data intensive problems, the middleware approach to Grid computing was developed in science laboratories where clusters distributed across the world were linked together in order to create Grids. The role of the Grid middleware in this paradigm was to ‘glue’ the clusters together to achieve interoperability. Notable Grid middleware include Globus , GLite and UNICORE [2].

The heterogeneity and growing nature of Grid has made it difficult for Grid System to have optimal working environment at all time. Nodes membership is dynamic, leading to suboptimal organization of Grids, network congestion and issue of load balancing which effects Node performance and availability. The Grid solutions so far are aimed towards fulfilling Grid standards such as GGF [3] standards in order to make it efficient but these approaches have some short comings. e.g., such a system lacks intelligence as most of the decisions from establishing and configuring to maintenance are done manually. For it to be more effective and adaptable such system approach should be focused towards autonomic Grid services hence user’s participation in many complex configuration and maintenance decision requiring extensive knowledge about the Grid would be minimized. Although these middleware have basic level of autonomy but the required focus to have autonomic attributes is still lacking.

2.3.1. Architecture of Grids

Perhaps the most important standard that has emerged recently is the Open Grid Services Architecture (OGSA), which was developed by the GGF. OGSA is an Informational specification that aims to define a common, standard and open architecture for Grid based Applications. The goal of OGSA is to standardize almost all the services that a grid application may use, for example job and resource management services, communications and security. OGSA specifies a Service-Oriented Architecture (SOA) for the Grid that realizes a model of a computing system as a set of distributed computing patterns realized using Web services as the underlying technology. Basically, the OGSA standard defines service interfaces and identifies the protocols for invoking these services. OGSA was first announced at GGF4 in February 2002. In March 2004, at GGF10, it was declared as the GGF's flagship architecture. The OGSA document, first released at GGF11 in June 2004, explains the OGSA Working Group's current thinking on the required capabilities and was released in order to stimulate further discussion. Instantiations of OGSA depend on emerging specifications (e.g. WS-RF and WS-Notification). Currently the OGSA document does not contain sufficient information to develop an actual implementation of an OSGA-based system. A comprehensive analysis of OGSA was undertaken by Gannon *et al*., and is well worth reading.

2.3.2. Standards in Grid Computing

There are many standards involved in building service oriented Grid architecture, which form the basic building blocks that allow applications execute service requests. The Web services based standards and specifications include, Program-to-program interaction (SOAP, WSDL and UDDI) [4], Data sharing (eXtensible Markup Language – XML), Messaging (SOAP and WS-Addressing), Managing resources (WS-RF or Web Services Resource Framework), Handling metadata (WSDL, UDDI and WS-Policy), Building and integrating Web Services architecture over a Grid, Triggering process flow events (WS-Notification).

As the aforementioned list indicates, developing a solid and concrete instantiation of OGSA is currently difficult as there is a moving target – as the choice of which standard or specification will emerge and/or become popular is unknown. This is causing the Grid community a dilemma as to exactly what route to use to develop their middleware.

2.4. Autonomic Computing

In this section, autonomic computing is introduced, e.g. why it is needed, what kinds of features an autonomic system has, how to apply autonomic computing to the Grid and what kinds of benefits it can bring to the Grid. Finally, some current works on autonomic computing are reviewed. Broadly speaking, autonomic computing refers to an infrastructure that automatically adapts to meet the demands of the applications that are running in it. Autonomic computing is a self-managing computing model named after, and patterned on, a human body's autonomic nervous system. An autonomic computing system is one that is resilient, and able to take both pre-emptive and *post facto* measures to ensure a high quality of service with a minimum of human intervention, in the same way that the autonomic nervous system regulates body systems without conscious input from the individual. The goal of autonomic computing is to reduce the complexity in the management of large computing systems such as the Grid. The Grid needs autonomic computing for following reasons. Complexity as Grid is complex in nature because it tries to couple large-scale disparate, distributed and heterogeneous resources – such as data, computers, operating systems, database systems, applications and special devices – which may run across multiple virtual organizations to provide a uniform computing platform. And the Grid is a dynamic computing environment in these resources and services can join and leave at any time.

2.4.1. Features of autonomic computing systems

A system that is to be classified as an autonomic system should have the these major features. It should be self-protecting system so that it can detect and identify hostile behaviour and take autonomous actions to protect itself against intrusive behaviour. Self-protecting systems, as envisioned, could safeguard themselves against two types of behaviour: accidental human errors and malicious intentional actions. To protect themselves against accidental human errors, e.g. self-protecting systems could provide a warning if the system administrators were to initiate a process that might interrupt services. To defend it against malicious intentional actions, self-protecting systems would scan for suspicious activities and react accordingly without users being aware that such protection is in process. Besides simply responding to component failure or running periodic checks for symptoms, an autonomic system will always remain on alert, anticipating threats and preparing to take necessary actions. Autonomic systems also aim to provide the right information to the right users at the right time through actions that grant access based on the users' roles and pre-established policies. It should be have self-optimizing components, to dynamically tune themselves to meet end-user or business needs with minimal human intervention. The tuning actions involve the reallocation of resources based on load balancing functions and system run-time state information to improve overall resource utilization and system performance. Self-healing is another ability of a system in which system recovers from faults that might cause some parts of it to malfunction. For a system to be self-healing, it must be able to recover from a failed component by first detecting and isolating the failed component, taking it off line, fixing and reintroducing the fixed or replacement component into service without any apparent overall disruption. A self-healing system will also need to predict problems and take actions to prevent the failure from having an impact on applications. The self-healing objective must be to minimize all outages in order to keep the system up and available at all times. The system should be self-configuring because installing, configuring and integrating large, complex systems is challenging, time consuming and error-prone even for experts. A self-configuring system can adapt automatically to dynamically changing environments in that system components including software components and hardware components can be dynamically added to the system with no disruption of system services and with minimum human intervention. Autonomic system should be based on open standards and provide a standard way to interoperate with other systems. An autonomic system should be integrated with a machine learning component that can build knowledge rules based on a certain time of the system running to improve system performance, robustness and resilience and anticipating foreseeable failures.

2.3.2 Basic working of Autonomic System

The Figure 2.1 shows how the autonomic system works in Grid (Plant). Basically it is divided into two parts manager and managed element. Manage element is Grid while manager includes sensor to expose the Grid information which analyzer processes with help of planner to decide the action to be taken and Knowledge Base of information containing events with corresponding action to be taken . Then comes policy which is the only component which interacts with the human, it is where human define the criteria for a action to be performed and finally the actuator which operates on the Grid with action selected.

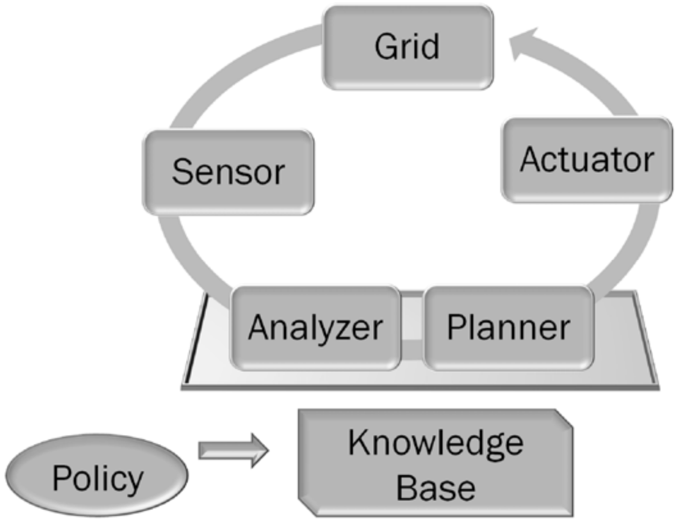


Figure 2.1 Autonomic Element Compositions

2.5. Summary

This chapter gave the basic idea about Grid computing and autonomic computing and its characteristics.

C h a p t e r 3

3. Architectural Design

3.1. Introduction

In this chapter different architectural diagram are shown describing ARDIG from different prospective. Starting from the high level architecture, ERM to the low level diagrams showing the Class details have been explained with visual aids. The chapter also contains the integration details with Phantom OS.

3.2. Architecture of ARDIG

Figure 3.1 shows different components in three different types of nodes. GridNode is responsible for SuperNode failure detection and mirroring its vital information. It provides the resource Broker information for SuperNode to aid with discovery service. SuperNode hosts the heartbeat component and local discovery service within the subGrid. Heartbeat component determines the availability of the GridNodes and increments the mean availability factor of the GridNode. Regional relays all the communication among the subGrids by providing Global Discovery service, organizing and optimizing services.

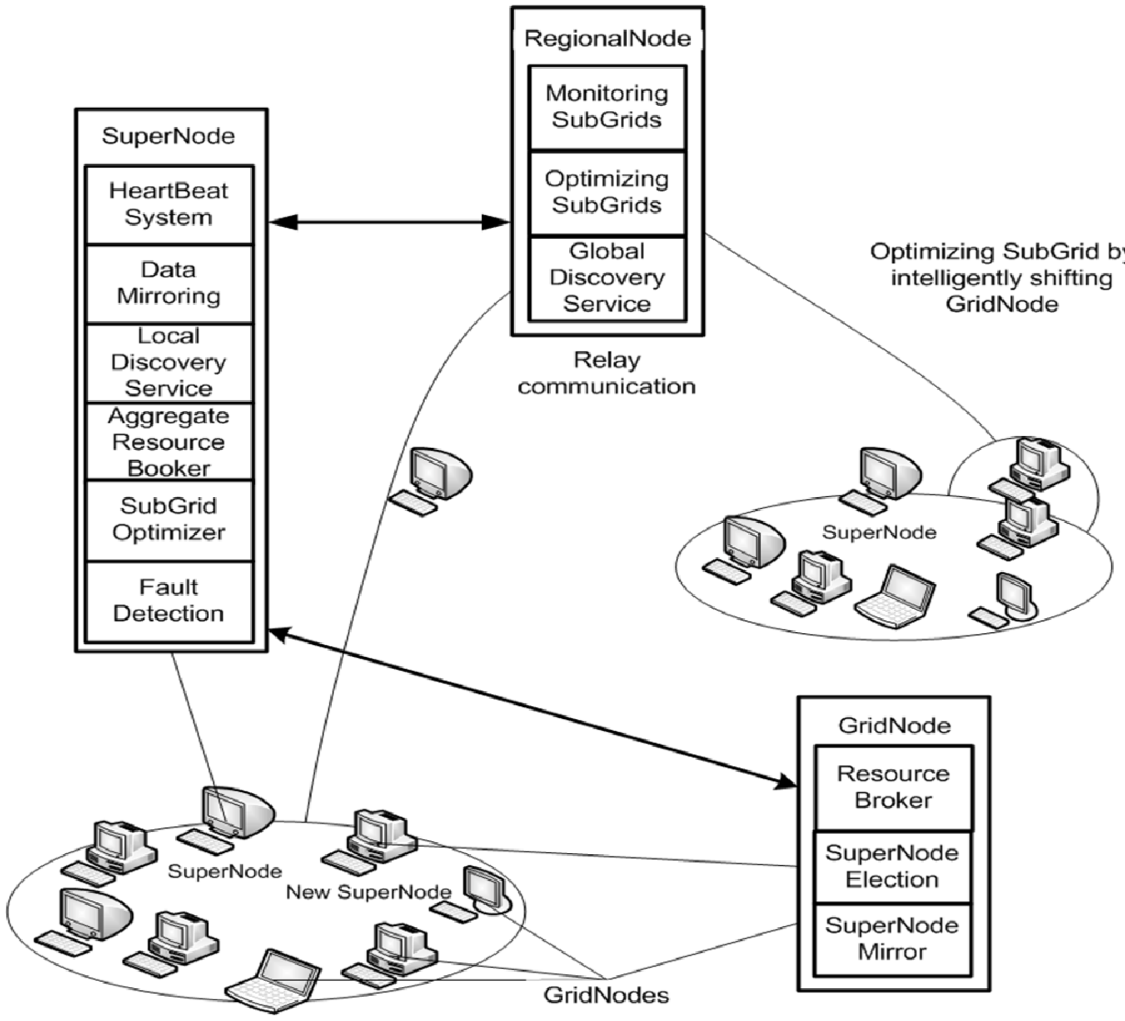


Figure 3.1: Proposed System Architecture

3.3. Benefits of proposed system

The proposed solution is to aid the discovery services in its decision making process by providing autonomic Grid service which would heal, organize and optimize the Grid System as depicted in the Figure. It could become part of any existing Grid System and would be interoperable with other Grid System. It would provide the following benefits; the existing grid infrastructure would benefit from it by having adaptable discovery services to its environment. This discovery service would be able to not only discovery the resources for any Grid application it could also manage the Grid system which include healing the network if any faults occur and organize the Grid system in order to optimize the performance of Grid, Another promising benefit would be to Grid OS [8]. Grid OS have reduced the complexity of setup and configuring the Grid system on any node by self configuring some of the decision which users would have been given. Example of such OS is phantom Os . This proposed solution would be added as an extra layer of modules to the Grid and by using this autonomic Grid Service better Quality of Service for Grid application can be provided like Intel built into its Itanium 2 [10] processor features of autonomic computing called the Machine Check Architecture (MCA). The MCA is an infrastructure that allows systems to continue executing transactions as it recovers from error conditions, it has the capability to analyze data and respond in a way that provides higher overall system quality therefore such services would reduce response time, number of node failure or availability and would make more efficient use of resources.

3.4. ERM of ARDIG

The figure 3.2 below is the ERM diagram of the ARDIG database. It shows the entities which include Regional Peer, SubGrid, SuperPeer, GridNodes, Job, User

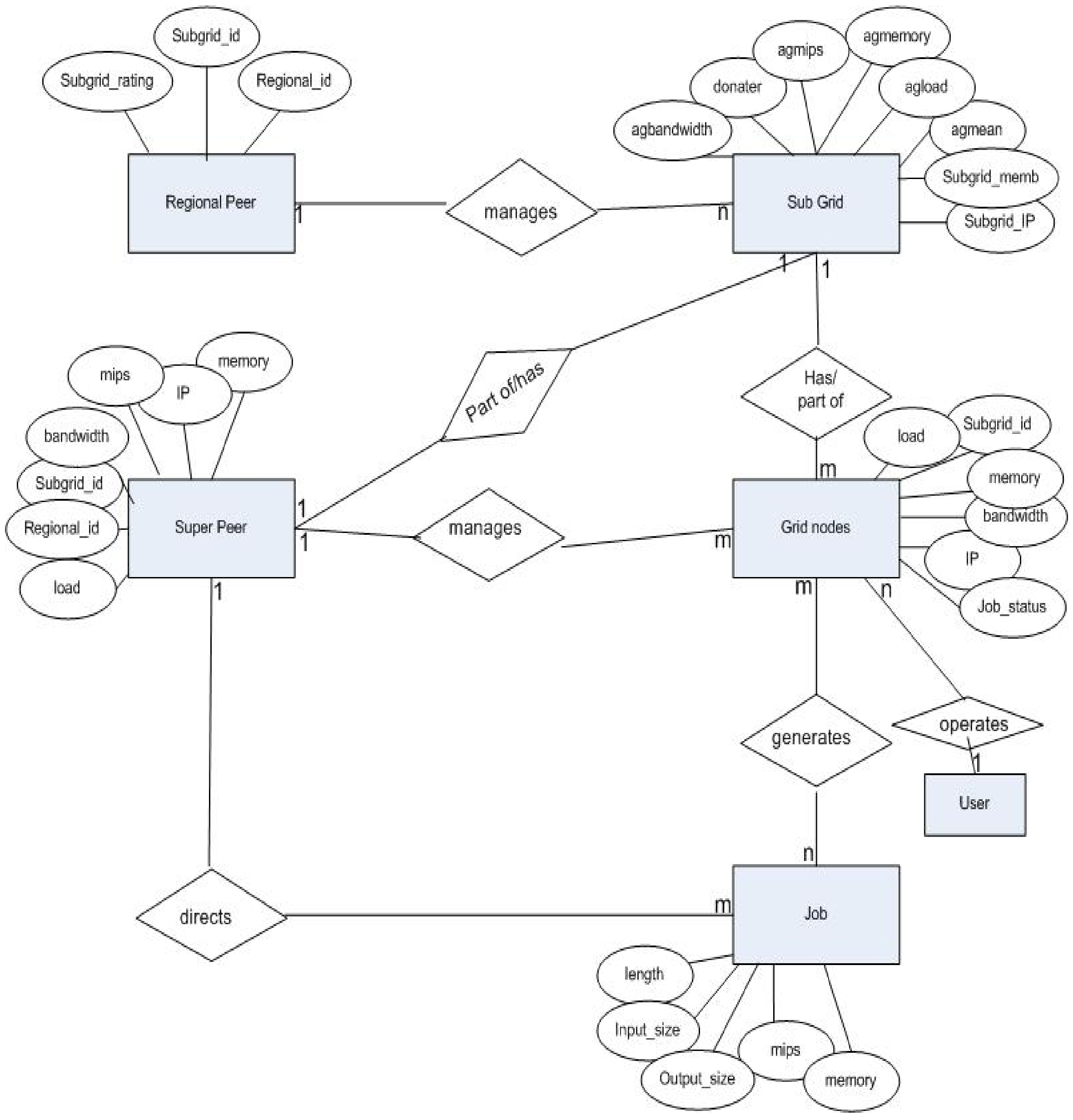


Figure 3.2: ERM of ARDIG

3.5. Class Diagram

The figure 3.3 below is the Class diagram of the ARDIG database. It shows the classes which include Regional Peer, SubGrid, Super Peer, GridNodes, Job, User and basically shows the web service related classes and also the method implementation.

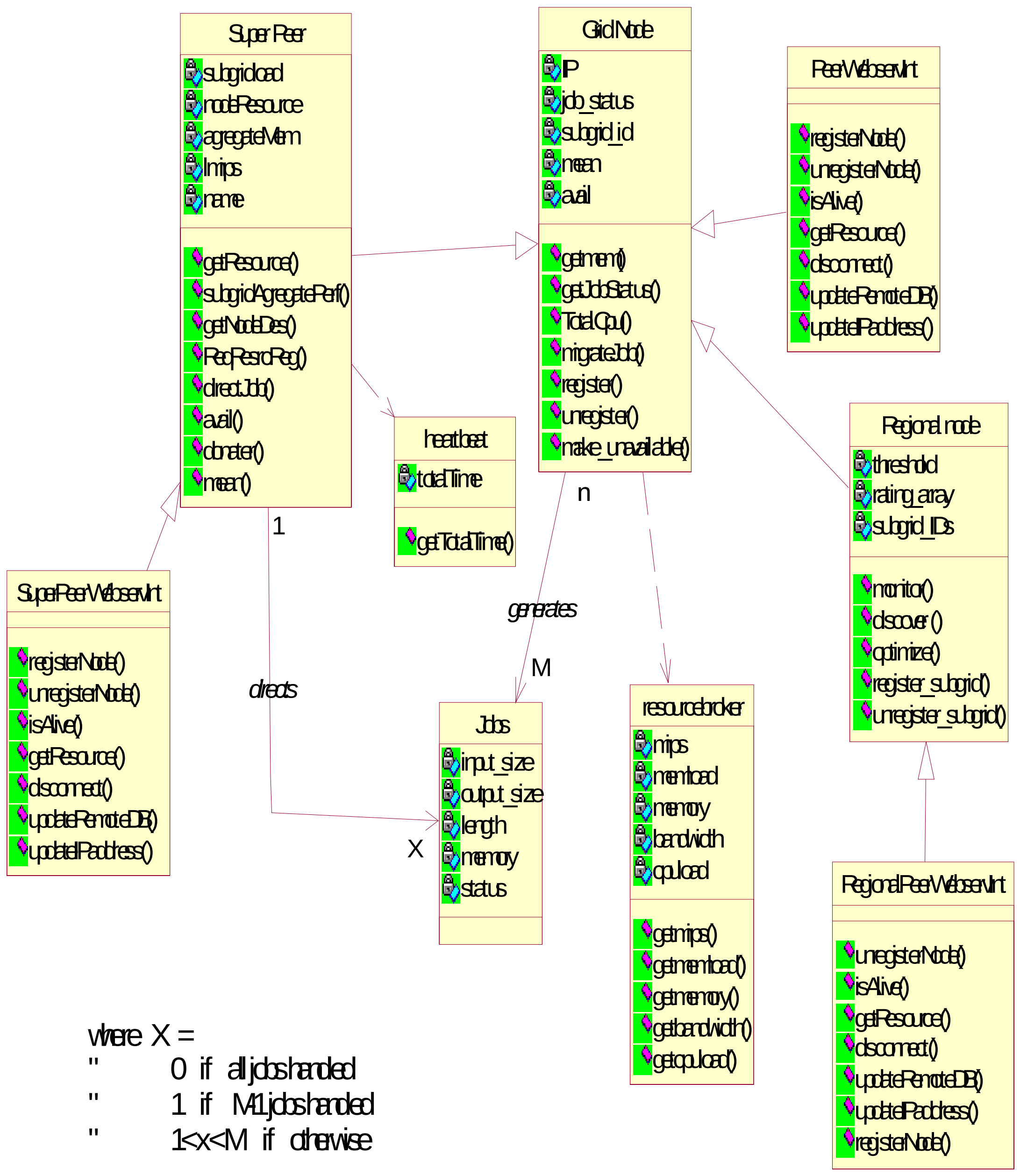


Figure 3.3: Class Diagram of ARDIG

3.6. Integration with Phantom OS

The Figure 3.4 below shows a sample architectural interaction between Grid OS components when the user submits a job. The Autonomic Discovery Infrastructure module basically provides an additional layer of abstraction which aids in managing; organizing and optimising the node organization by provide vital information and functions.

The user executes an application and the operating system’s Grid Enabled Process Management system automatically detects that this is a Grid enabled job and creates its execution threads, which it then forwards to the Resource Brokering and Scheduling engine where the threads are moved for processing to nodes that are more powerful than the user’s own node. The communication between the nodes takes place via the P2P subsystem, which is also responsible for the discovery of resource while the autonomic discovery infrastructure layer provides a virtual link among each node.

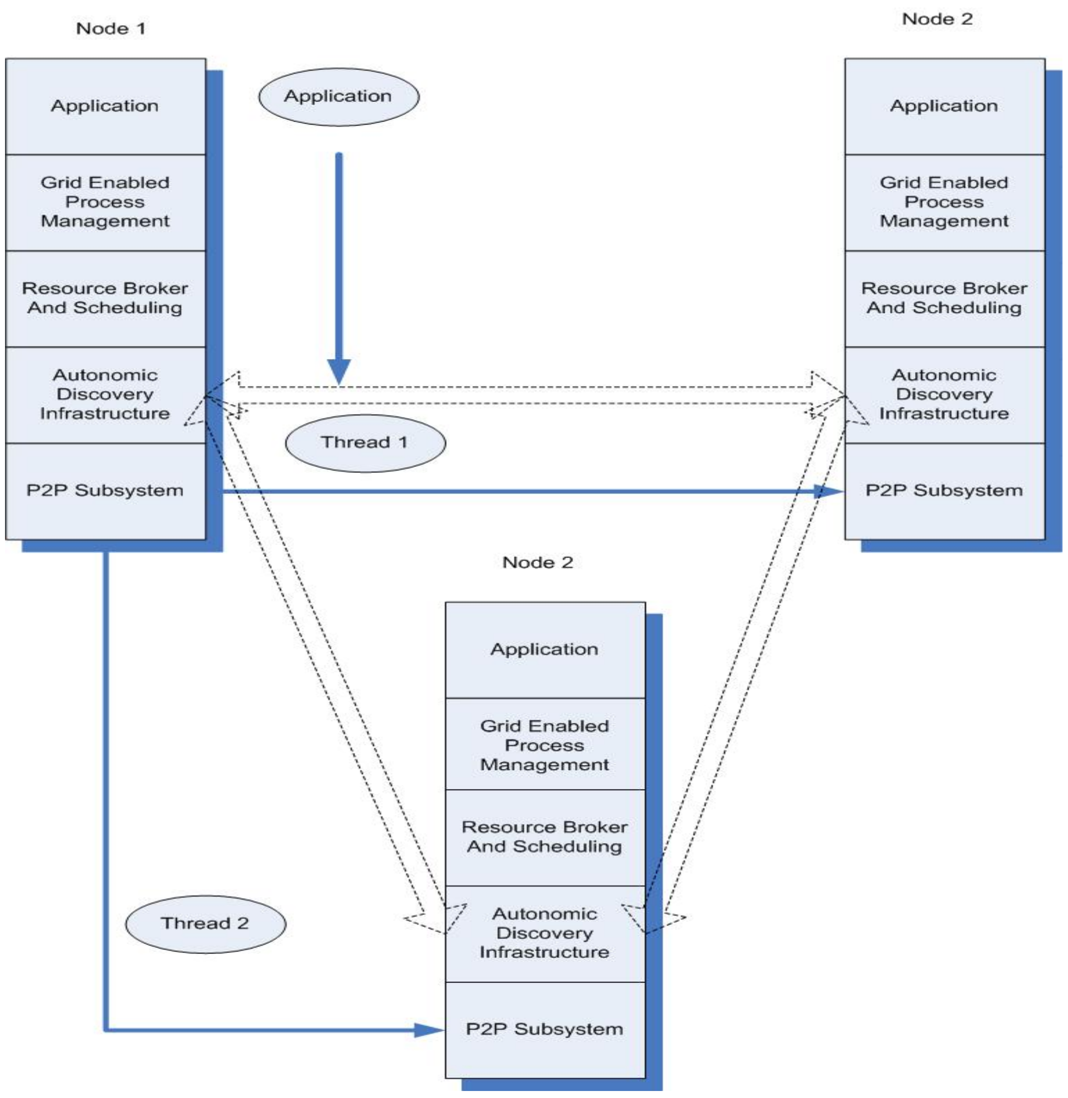

Node 1
Application
Grid Enabled Process Management
Resource Broker And Scheduling
Autonomic Discovery Infrastructure
P2P Subsystem
Application
Thread 1
Thread 2
Node 2
Application
Grid Enabled Process Management
Resource Broker And Scheduling
Autonomic Discovery Infrastructure
P2P Subsystem
Node 2
Application
Grid Enabled Process Management
Resource Broker And Scheduling
Autonomic Discovery Infrastructure
P2P Subsystem

Figure 3.4: Integrated System Architecture

The ARDIG project comprises a major portion in the Middleware level and resource Broker part of the User space in the Phantom OS [11] Architecture.

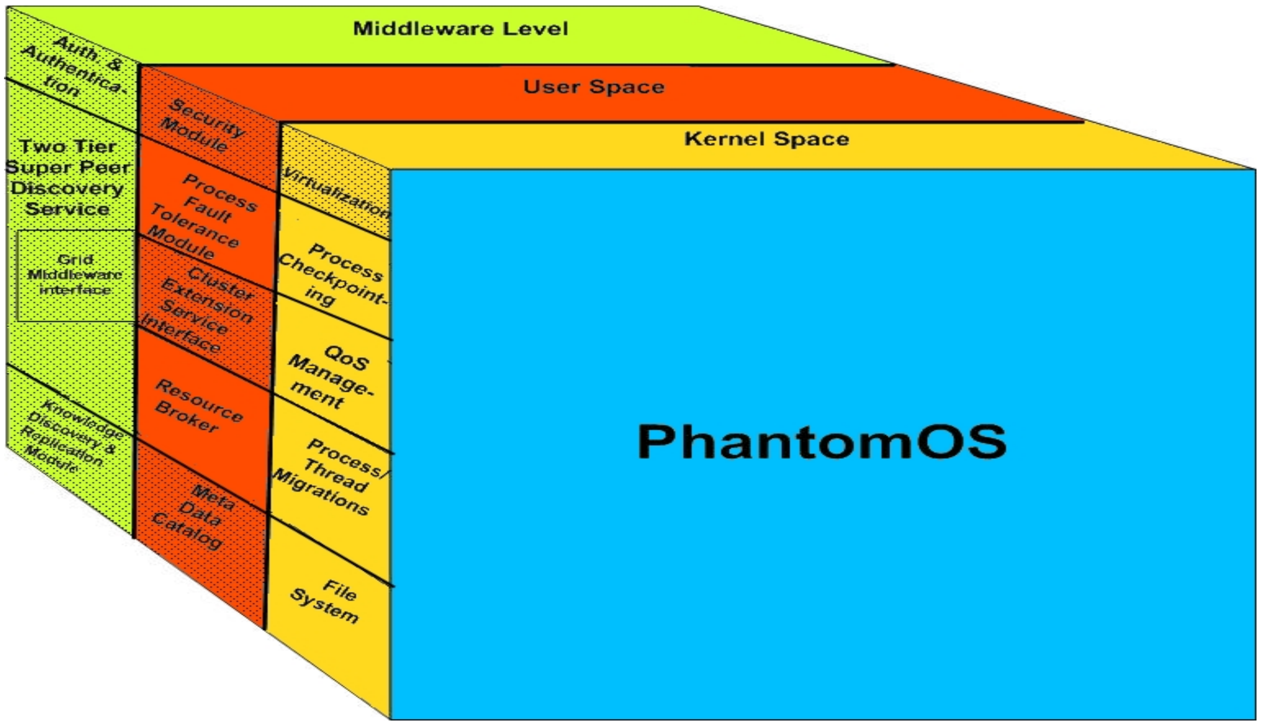

Middleware Level
User Space
Kernel Space
Auth. & Authentica-tion
Two Tier Super Peer Discovery Service
Grid Middleware interface
Knowledge Discovery & Replication Module
Security Module
Process Fault Tolerance Module
Cluster Extension Service Interface
Resource Broker
Meta Data Catalog
Virtualization
Process Checkpoint-ing
QoS Manage-ment
Process/ Thread Migrations
File System
PhantomOS

Fig 3.5 Phantom OS Architecture

In figure 3.5 the PhantomOS can be seen from two perspectives: An integrated Grid Stack to allow for rapid deployment of Grids, while making administration of Grids easy. And as an Operating system which provides built in support for Grid computing. The components which have a dotted background show those components which are relevant to PhantomOS as a Grid Stack others are for PhantomOS as a complete Operating System. There is overlap between both modes. For example the Super Peer module is used for both for PhantomOS as an OS and PhantomOS as a Grid Stack. PhantomOS is designed after a modular paradigm. Kernel changes can be turned off by unloading the appropriate kernel modules. If an organization chooses to use the stack configuration it can easily unload the kernel space modifications and use Grid computing from a user and middleware level.

As related to the project Scope the following modules have been implemented in ARDIG; Discovery Service, Self Healing of Grid, Self Organization of SubGrids, Resource Broker, Heartbeat Monitoring, Discovery Service

3.7. Summary

This chapter gives us the understanding about the architecture of ARDIG, its link with Phantom OS and what components of ARDIG which were implemented as part of the Degree Project. The following chapters will describe each component in more detail.

Chapter 4

4. Discovery Service

4.1. Introduction

The discovery scheme is an enhancement over P2P discovery and centralizes discovery technique. The enhancements target certain limitations, primarily dealing with the adaptability of the algorithm to hybrid Grids and limiting the overhead of communication between the nodes in a single instance of resource discovery and usage. Certain enhancements deal with limiting the potential for all–to-all communication which plague existing peer to peer networks. There are many Grid Resource discovery mechanisms [12, 13, 14, and 15] but none of them is network topology-aware [16].

4.2. Two Tier Architecture

Here two-tier based super peer architecture is introduced: the lowest tier is a machine level granularity sub-grid, which consists of machines that have good network connectivity between them. Each sub-grid is represented by a super-peer, which is the most available machine within the vicinity of the sub-grid. At the top-most tier the granularity is in terms of sub-grids, and these are grouped into regions depending on geographical proximity of the super peers. The regions are represented by a region peer, as shown in Figure 4.1. A virtual organization (VO) in this system can be at any level: it can consist of individual machines or be an aggregation of entire subgrids or of entire regions. Interactive applications will be handled at a machine-level VO, whereas large-scale Grid applications will require aggregations of entire sub grids.

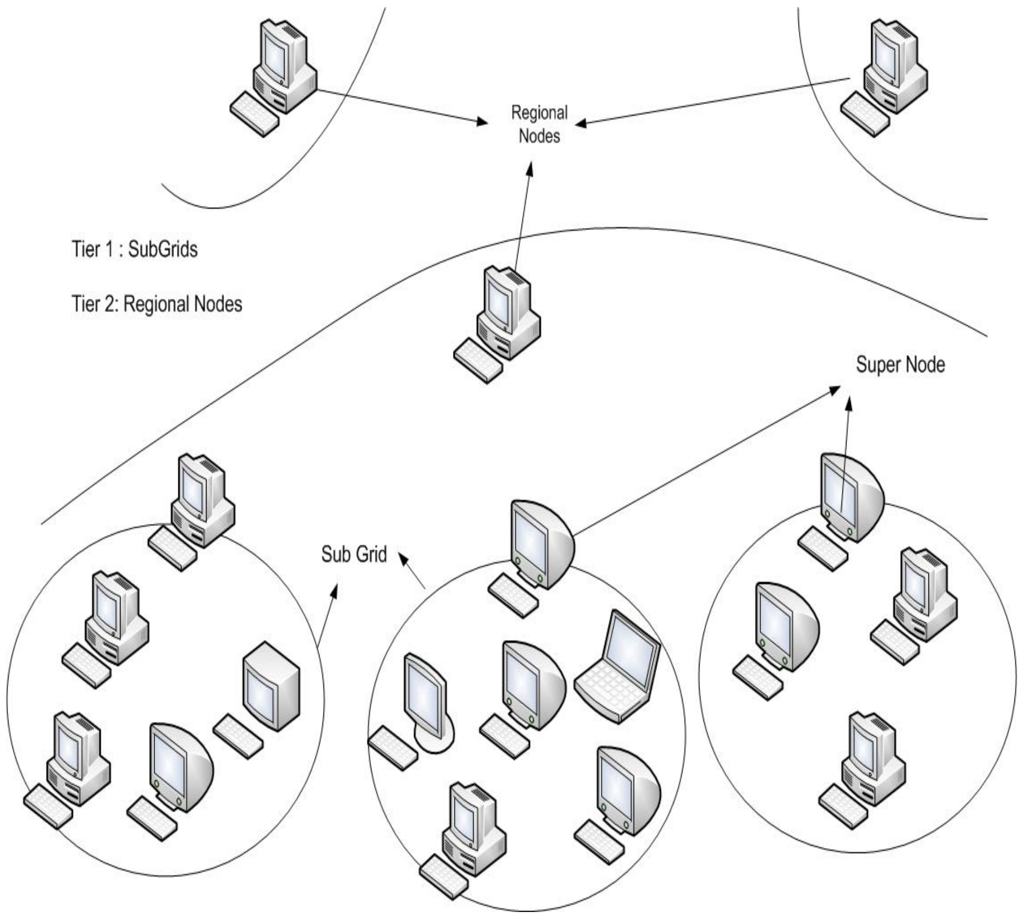


Figure 4.1 Two tier super peer architecture

The whole concept of two-tier super peer based system was developed for three main reasons; to improve the network usage, by allowing a resource request to propagate to peers in close proximity, thus limiting overall network traffic, and improving response latency, to improve the quality of results, by propagating the request until a suitable resource has been found, while limiting the network traffic as much as possible; to provide a scalable and efficient framework for Grid OS, by dynamically grouping nodes into sub-grids, and clustering sub-grids into regions, QoS is ensured for individual nodes, and the overall network efficiency is enhanced by limiting the flow of resource requests and to enable the creation of different kinds of Grids, as required in different domains, from simple cluster oriented Grids of today to the ad-hoc Grids relevant to common users and businesses resource discovery mechanism will be explained separately in terms of tiers.

4.3 Sub Grid level Resource discovery

The sub-grid is analogous to a cluster of computers, and is the lowest tier in the system. Resource discovery and brokering is done internally in the sub-grid in a semi-centralized fashion. The central server in the sub-grid is the super peer, which corresponds to the most available machine in the cluster, and has the responsibility of handling, managing requests and providing a registration interface to new nodes. Upon joining a sub-grid members register their presence with the super peer. When a node of a sub-grid needs a resource, it sends a request query to its super-peer which returns the list of resources matching the user's query constraints, if matching resources are available. If the super-peer cannot satisfy the request, it then forwards the query request to the region peer. Once the requesting machine has a list of the machines within the sub-grid it contacts each in a P2P fashion and the resource broker determines the suitability of the discovered nodes to execute the user application, leading to eventual migration of the job.

4.4 Region Level Resource discovery

If a resource request cannot be satisfied from within the subGrid, the region peer comes into play. The region peer has a notion of the cumulative power of a sub-grid, and based on it takes a decision on which sub-grids have the required resources to compute the job. The cumulative power of a subGrid is determined by aggregating individual resource descriptions and calculating a theoretical peak. When such sub-grids are found, the job request is forwarded to them and then the resource brokering and scheduling process takes place within the new sub-grid. If the region cannot satisfy the resource requirements it then contacts other regions in a P2P manner.

The figure 4.2 shows how the discovery service registers the node in with the super peer.

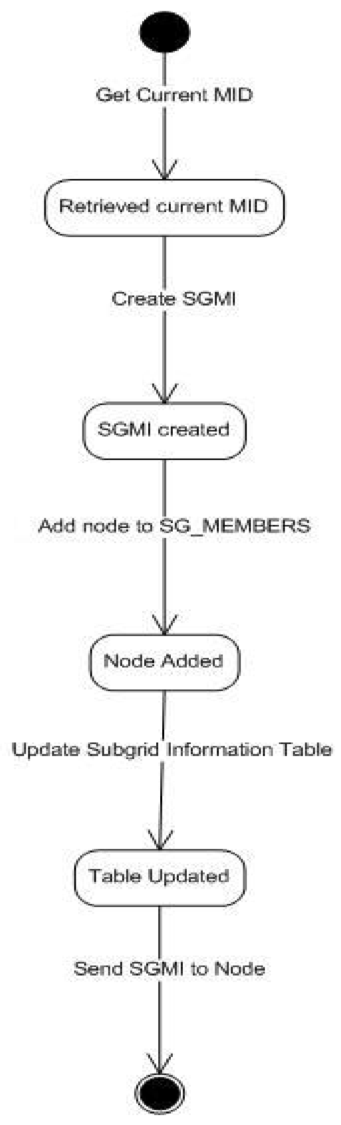


Figure 4.2 Discovery Service Registration Process

The figure 4.3 shows the process in which the discovery service performs the signing in operation when ever node comes online.

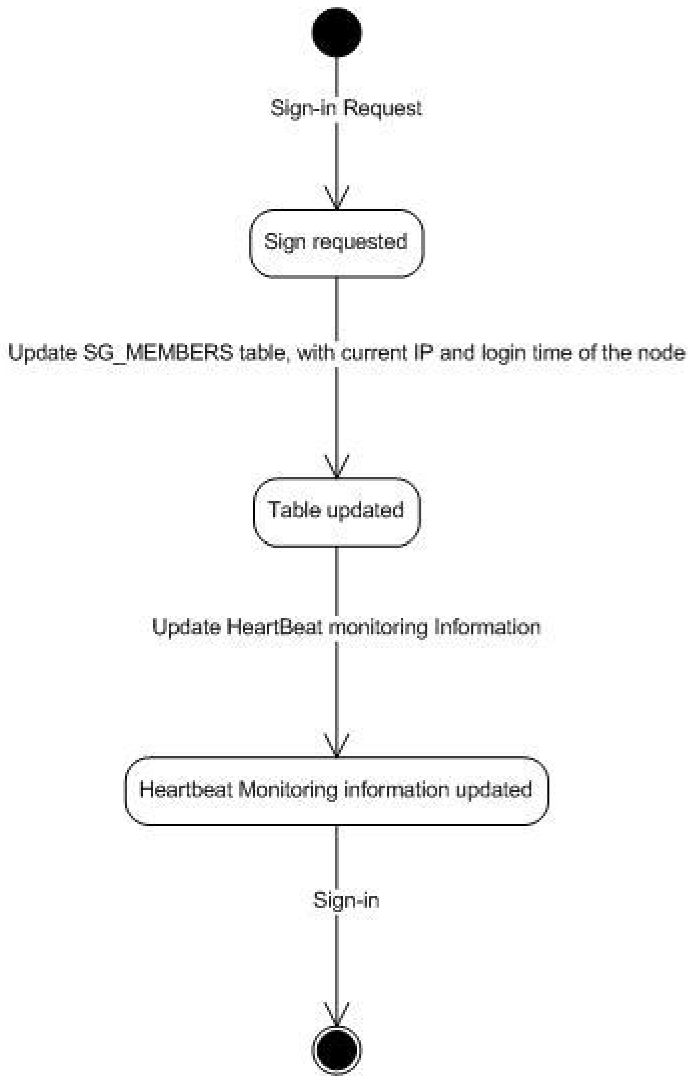

Sign-in Request
Sign requested
Update SG_MEMBERS table, with current IP and login time of the node
Table updated
Update HeartBeat monitoring Information
Heartbeat Monitoring information updated
Sign-in

Figure 4.3 Discovery Service Sign in Process

The figure 4.4 below shows how the Grid Node is signed in with the super peer. Basically the heart beat is responsible for determining the availability of the Grid Node as soon as it detects the presence of Grid Node it tags the avail attribute of the node in sg_members table in database to 1 showing its availability.

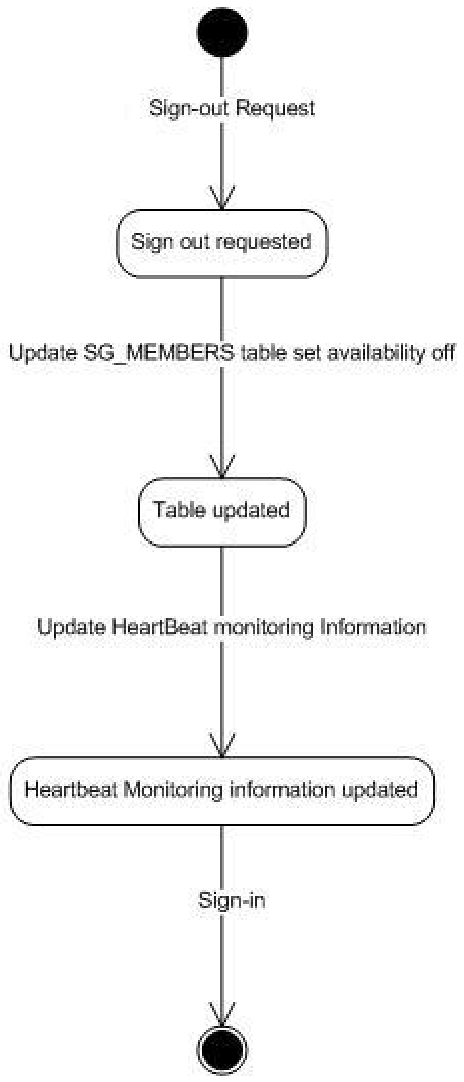

Sign-out Request
Sign out requested
Update SG_MEMBERS table set availability off
Table updated
Update HeartBeat monitoring Information
Heartbeat Monitoring information updated
Sign-in

Figure 4.4 Discovery Service Sign out Process

The discovery service resource request comprises of two major steps. One in which the resource is looked up in the subGrid which is simpler as it looks in the sg_members table and finds any node which is available and matches the requirement. This is also described in the figure below.

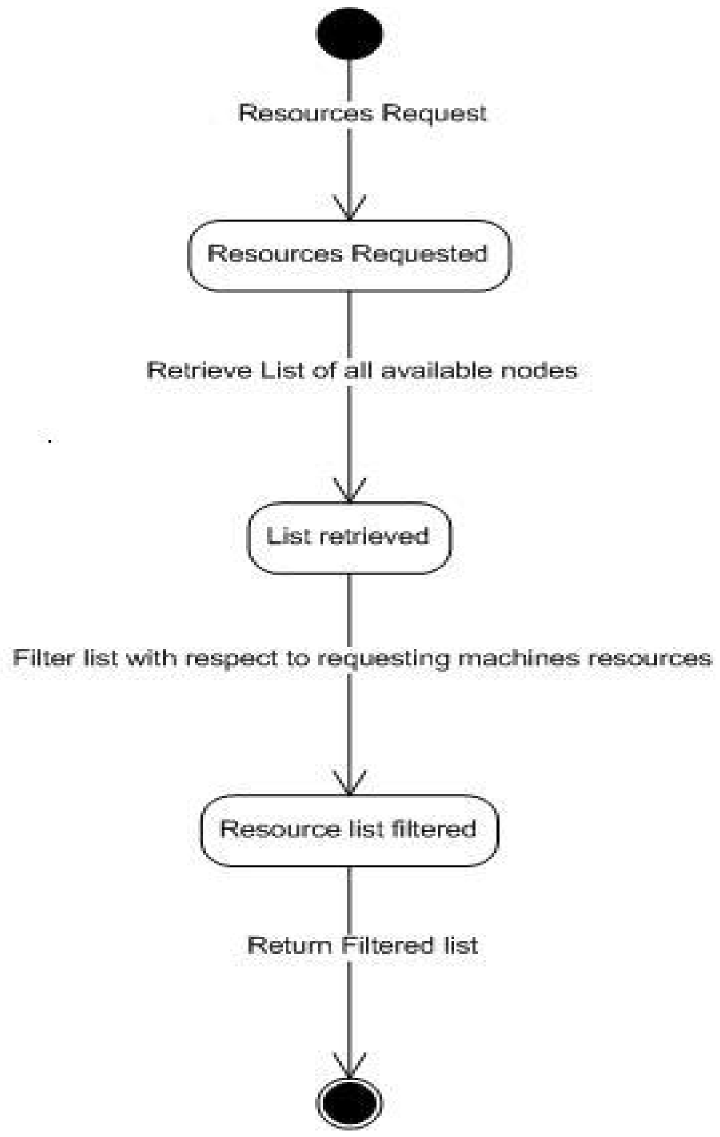

Resources Request
Resources Requested
Retrieve List of all available nodes
List retrieved
Filter list with respect to requesting machines resources
Resource list filtered
Return Filtered list

Figure 4.5 Discovery Service Resource Request Process

4.5. Summary

This chapter includes detail about discovery service, different level of different service and the registration, signing in and out process.

Chapter 5

5 Autonomic Resource Discovery Framework

5.1. Introduction

The Autonomic Resource Discovery Infrastructure builds on the Grid system and OGSA and extends Discovery services to support autonomic behaviour. This component operates in parallel with other components. Its specialized discovery service should be used to employ autonomic services. The purpose of this component is to provide autonomic infrastructure to the existing grid system.

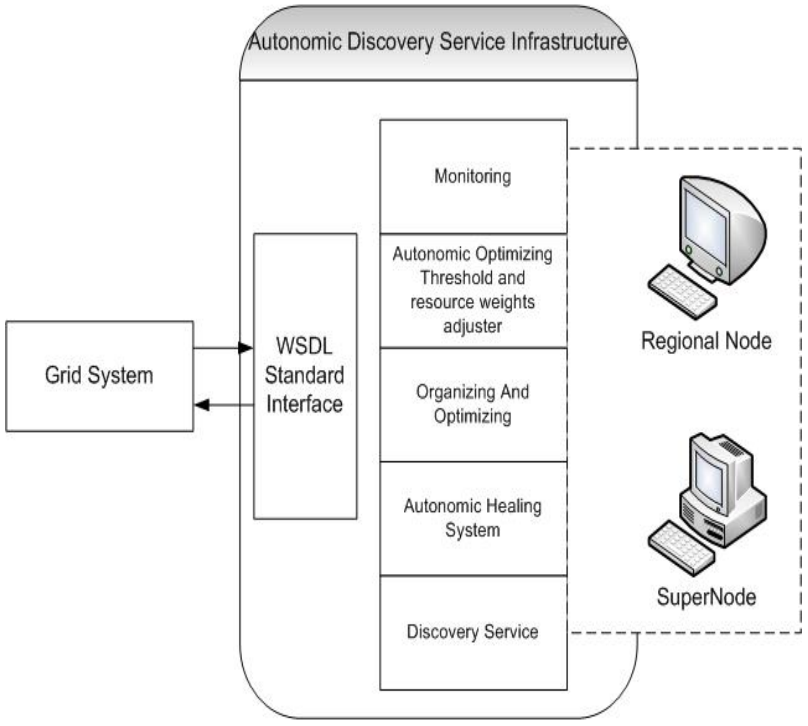

Autonomic Discovery Service Infrastructure
Monitoring
Autonomic Optimizing Threshold and resource weights adjuster
Organizing And Optimizing
Autonomic Healing System
Discovery Service
WSDL Standard Interface
Grid System
Regional Node
SuperNode

Figure 5.1: Architecture Overview of Autonomic Resource Discovery Framework

Figure 5.1 shows different components, the autonomic component which is optimizer and healing system are based on statistic figures relating to resource usage, deficiency, reputation, performance and frequency of changes related to discovery information to determine usage pattern which would be used to adjust the threshold and weights used in organizing and healing the subGrids.

The important thing is that the design of this infrastructure should be customizable to replace its components or to add new components. So that if any new components like to put into the equation some other information about node then this component should be plugged to this module without causing any chain of reaction to accommodate these changes. For example in order to cater to network unreliability it was decided to modify the delivery semantics of any remote invocation so that such functions could be added with ease.

5.2. Self Healing

The basic idea behind this feature is to provide healing of any damage that could occur in the Grid system. For example nodes could crash, network congestion and super peer failure. The proposed infrastructure would provide fault tolerance to such problems and also would have the element to deal with unexpected faults. In the following two figures 5.2 and 5.3 a failure scenario is shown which shows what are the action taken as soon as the super peer goes down and the next figure shows the organization of node of electing new super peer. In this case the autonomic self healing part of the infrastructure would have to do two things to re-establish the system. One is to record the information that is lost by the super peer and the other is to elect new super peer from the rest of the peers in the organizations

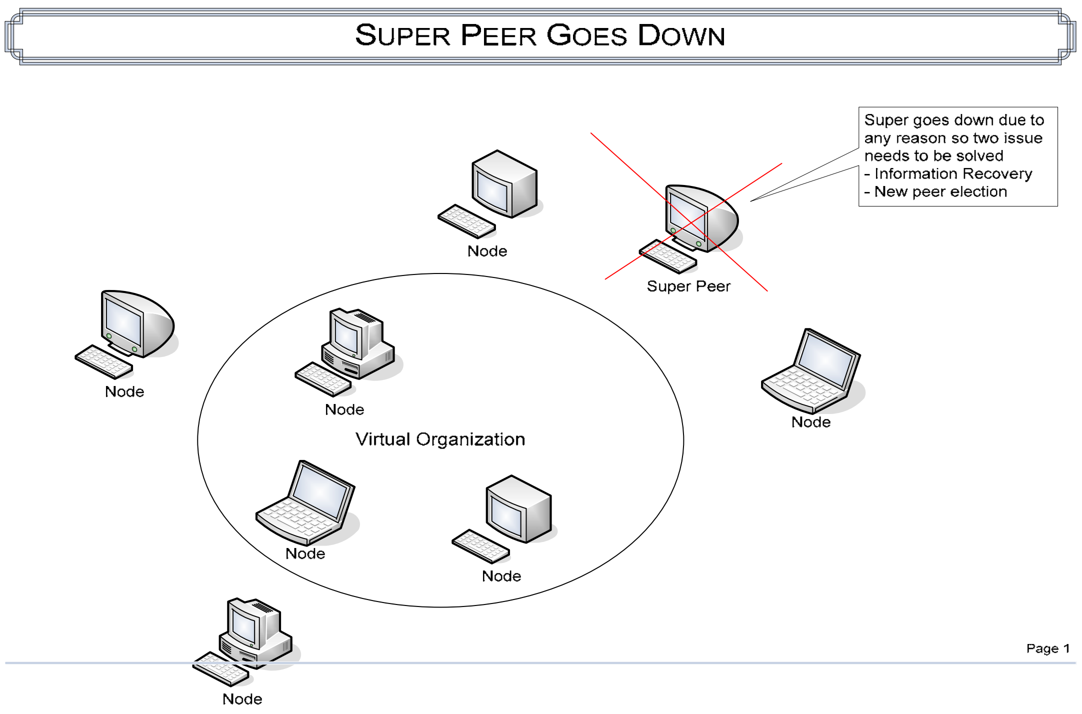


Figure 5.2: Scenario where super peer goes down.

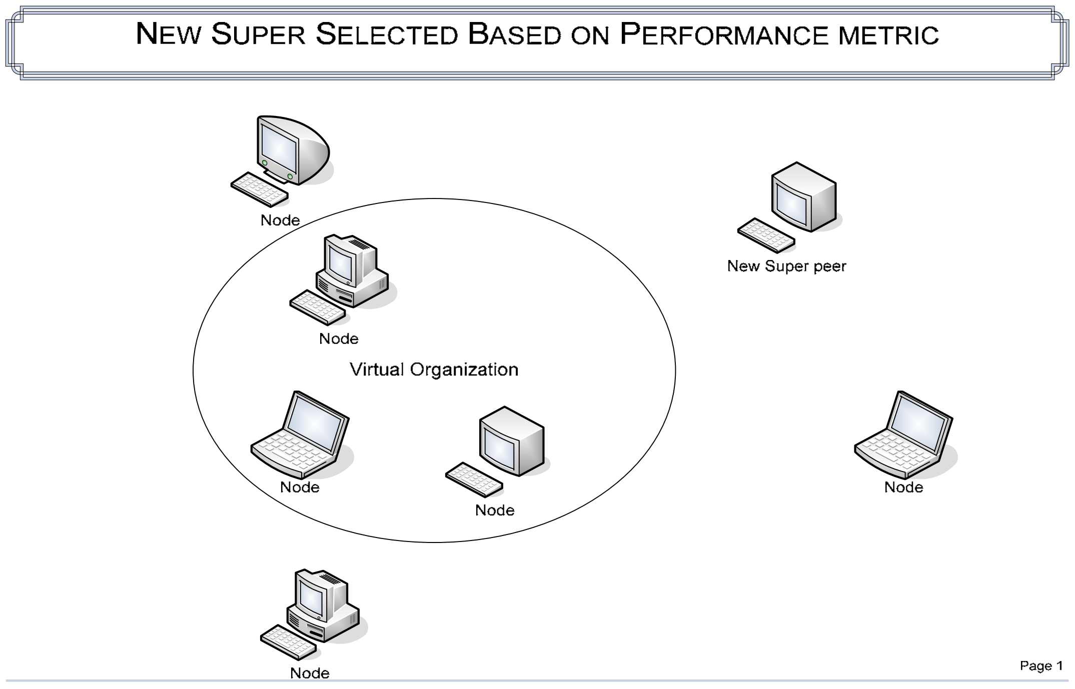


Figure 5.3: Scenario where new super peer is elected

The concept behind this is to mirror the data in the Super Node among N no of Grid Nodes. Super Node would be the primary node while the rest of the N nodes are secondary node. In case primary node malfunction then any one of the N backup nodes is upgrade to primary node avoiding any loss of information and delays with the operation current executing.

The self Healing algorithm given in figure 5.4 below shows the procedure how nodes are elected.

Primary node and secondary node approach

Primary node is the current active SuperNode

Secondary node are backup SuperNode

Mirroring data among Secondary

Election Criteria

Ideally node with best performance and less load and higher online probability among secondary node.

Such nodes can be acquired from other SubGrid. In such **case** there would be some delay as it has to updated with the subGrid information.

Figure 5.4: New super election algorithm

5.3. SubGrid Workload Prediction

Prediction of workload is vital for optimizing the subGrid as it aids to take proactive measure in case subGrids expect heavy loads in future. SubGrid workload depends upon no of factors [19]. Like the amount of free resource available and the rate at which jobs are generated. In simple scenario the subGrid optimization factors 'H' depends upon H=(N*C-R*P) where N is no of jobs generated, C is cost of processing job, R is the no of resource available, P is the free of resource available. But in reality resource would not be available at all time; the current rate of job generation wouldn’t be enough to predict future loads and estimation of cost for jobs is also another issue. So keeping in mind these issues the more appropriate equation to represent this factor is as follows:

$$H= £\,(Rate*(JR*L))-£(R*M*(RS-RL)) \quad ....\ 5.1$$

Where JS is the job resource requirement and L is job Length, M is the mean active time and RS is resource capability and RL is the load on the resource.

5.4. Self Organizing

System would facilitate the scheduling and execution of Grid applications by aggregating Sub Grids into a single virtual Sub Grid, electing a leader which would manage the scheduling and eventual execution. This part of the proposed idea is also concern with the reorganization of the virtual organization among the nodes. This reorganization would optimize the setup by readjusting the nodes such that the load would be balanced among the nodes. Sub Grids which are idle or have low activity would be selected to donate node to other Sub Grid. Figure 5.5 depicts the Scenario where external node is added to the Sub Grid.

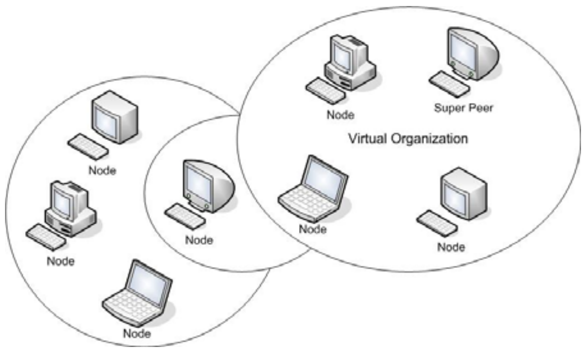


Figure 5.5: External node addition to the Sub Grid.

5.5. Optimizing Process

The aim is to organized nodes such that each subGrid has enough resources with respect to the history of job loads it carried. The figure 5.6 below is a sequence diagram showing how the optimization is conducted.

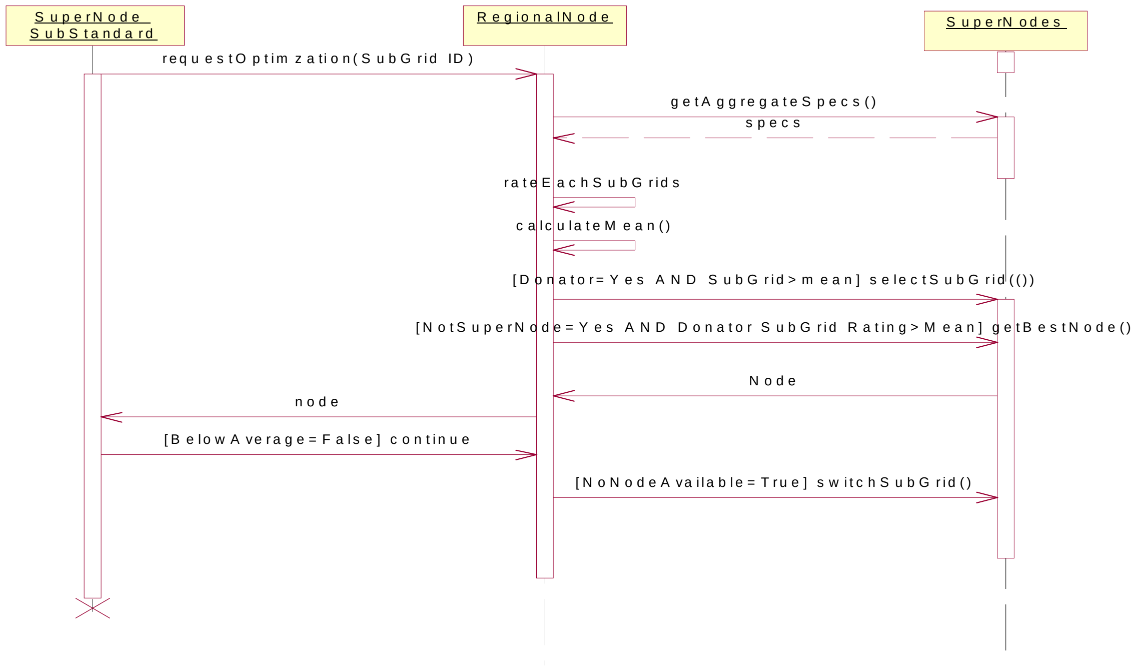

SuperNode SubStandard
RegionalNode
SuperNodes
requestOptimzation(SubGrid ID)
getAggregateSpecs()
specs
rateEachSubGrids
calculateMean()
[Donator=Yes AND SubGrid>mean] selectSubGrid(())
[NotSuperNode=Yes AND Donator SubGrid Rating>Mean] getBestNode()
Node
node
[BelowAverage=False] continue
[NoNodeAvailable=True] switchSubGrid()

Figure 5.6: Organizations of Nodes and Shifting Procedure

There are two optimization strategies as follows; Shift higher free resource GridNode performance with less local nodes from donator subGrid and shift higher job generating node from target subGrid.

Let ‘Th’ be threshold as defined earlier and ‘recieverSubGrid’ be the subGrid which is been denoted a node and ‘denotorSubGrid’ is the subGrid which denotes node. Note the calculation of threshold and nodes rating are discussed in coming sections. Below figure 5.7 shows how the organization is done.

```
While  ( recieverSubGrid  < Th AND denotorSubGrid  > Th)
    selectedNode = max ( nodeRating ( denotorSubGrid,for all g ) )
    recieverSubGrid.add (selectedNode)
    denotorSubGrid.remove (selectedNode)
```

Figure 5.7: Algorithm for shifting nodes

The donator subGrid is the subGrid with the best rating. Note only the subGrid which wants to denote will participate. Node elected shouldn’t be Super Node as well. Each time shifting is done the rating of both subGrid is updated. If the donator subGrid would be lower than threshold after denoting then in this case the denote SubGrid would be shifted to the next subGrid with the best rating.

5.5.1 Node Assessment

Most vital issue during organization is comparing nodes proving which is better. To resolve this, the following formula in figure 5.8 has been used:

Node Rating = (weightCPU * free_Cpu + weightMemory * free_memory) +

weightBandwidth * Bandwidth + weightMean * meanActiveTime 5.2

The reason why free resource is chosen is because it is unbiased. For example a machine with high specs is not necessary ideal because it could have higher load too. This equation can be used to rate node as well as subGrid without making any changes.

5.5.2 Threshold determination

Threshold value is used during optimization to provide stoppage criteria. It stops from taking too many nodes from the donator subGrids and it also allows how much nodes would be enough to satisfy the receiver subGrid. The threshold value is given as in figure 5.9:

Threshold = sum (rating of subGrids)/No of subGrids 5.3

History of threshold would be archived in order to avoid optimization in case there is minor change in its value.

5.5.3 Weight calculation

Let **‘**agCPU**’** is the aggregate value of all the Grid Node processor power in all subGrids in contact with particular regional peer and **‘**agMemory**’** is the aggregate value of all the Grid Node total memory in all subGrids in contact with particular regional peer **‘**agBandwidth**’** is the aggregate value of all the Grid Node bandwidth in all subGrids in contact with particular regional peer and **‘**agLCPU **‘**is the aggregate value of all the Grid Node CPU load in all subGrids in contact with particular regional peer **‘**agLMemory**’** is the aggregate value of all the Grid Node memory load in all subGrids in contact with particular regional peer and **‘**agLBandwidth**’** is the aggregate value of all the Grid Node bandwidth usage in all subGrids in contact with particular regional peer and **‘**agMean**’** is the average value of all the Grid Node mean availability factor in all subGrids in contact with particular regional peer

‘10’ is used because it is being scaled from 1 to 10

Note: this formula would be used among all those subGrids which are in contact with particular regional peer. (No use of shifting nodes among subGrids in different regions) The reason for calculating weight is because in order to determine which Node is better, is that its specs, loads and availability has to be compared but during optimization the ideal node elected to shift is the one which match best with the requirement of certain characteristic of resource for the subGrid so these weights are used to make these characteristic visible during comparison. Figure 5.11 shows how this theory is being calculated

weightCPU= (agLCPU/agCPU)*10

weightMemory= (agLMemory/agMemory)*10

weightBandwidth=(agLBandwidth/agBandwidh)*10weightMean= (agMean)*10

.... 5.4

5.6. Summary

This chapter summaries the main component of ARDIG that is self healing and self organization, it contains the design and algorithm related to the components in detail.

C h a p t e r 6

6. Resource Broker

6.1. Introduction

A Resource Broker is a central component in a Grid computing environment. The purpose of a Grid resource broker is to dynamically find, identify, characterize, evaluate, select, allocate and coordinate resources with different characteristics most suitable to the user's submitted job. Most existing resource brokers require too much user intervention and involvement to operate, and these are designed for batch applications. Thus these Brokers are not feasible for the end user who is just concerned with the ease of use, and high response and minimum turnaround time of interactive applications, which are not supported by existing Grid resource brokers. Present desktop operating systems take brokering and scheduling decisions taking only the local resources into consideration.

The Grid computing environment is a dynamic environment where status and load on resources are subjected to changes. Hence in such kind of environment it is very complex for the Broker to predict the performance and efficiency of the application on particular given resource. This problem is being addressed by current Grid middleware through policy based scheduling, policy negotiation and advance resource reservation schemes. In this scenario the Broker has some kind of exclusive control over system's resources in order to improve its performance and decision making ability. For the Grid operating system a brokering and scheduling engine built upon the underlying OS kernel which takes entire pool of resources available across the Grid is envisioned.

A Peer to Peer resource broker framework, in which everything from matchmaking of requirements and available resources, down to the scheduling is done cooperatively with Peers, is proposed. Thus enabling compute intensive and memory intensive applications to make best use of the Grid resources in order to achieve high throughput. In the proposed framework each resource in the system will advertise their most recent status dynamically. The task of the Broker is to collect this information and select most optimum machine among the eligible machines based on the job's characteristics and requirements that is submitted by the user. The Job requirements, the computational demands of the application, will be specified in the SQL database. The resource broker will assume that this information is collected by the Job Analyzer (Estimation service) of the resource management system of Grid OS. Resource broker waits for the job to be submitted by the user through console. On the submission of the job the first step taken by the resource broker is to interact with the Job Estimation service to retrieve the job execution requirements of job constraints.

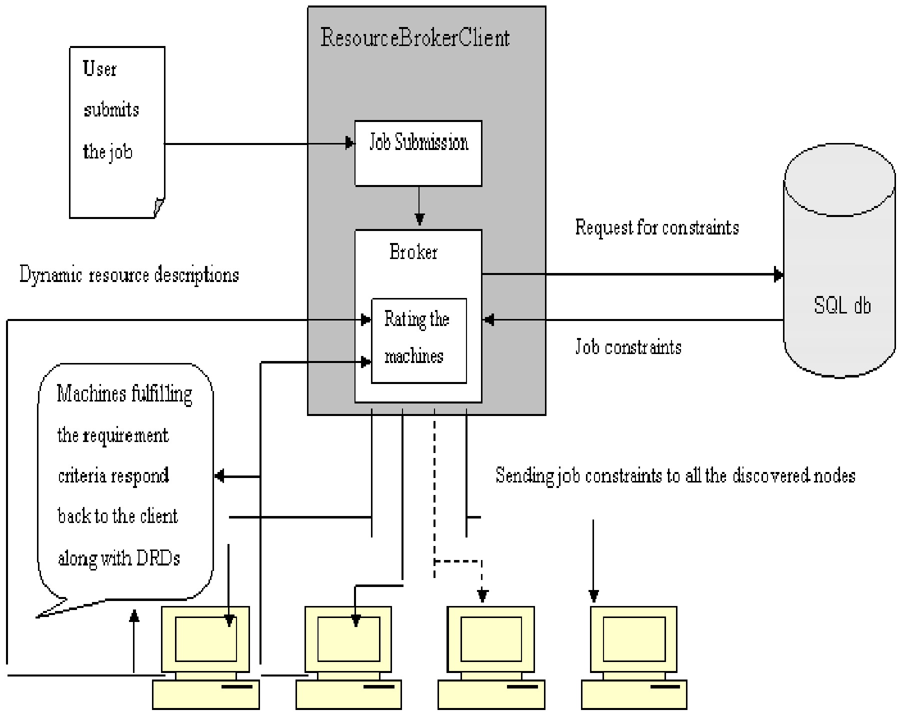

ResourceBrokerClient
User submits the job
Job Submission
Broker
Rating the machines
Request for constraints
SQL db
Job constraints
Dynamic resource descriptions
Machines fulfilling the requirement criteria respond back to the client along with DRDs
Sending job constraints to all the discovered nodes

Figure 6.1: Resource Broker architecture

Figure 6.1 shows the basic architecture and working of the resource broker in the Grid environment. Every authenticated machine in the Grid which is willing to provide its computation resources to other machines runs a resource broker service on top of its underlying operating system. Job is received by the resource broker client and it interacts with all other discovered set of machines in the P2P Grid environment, and each running resource broker service inside.

6.2. Resource Filtering

The first task performed by the Resource Broker server is to carry out resource filtering i.e. filtering out itself among the discovered set of machines if it does not fulfils the minimum requirement eligibility criteria. Therefore only those machines will respond back to the resource broker client which passes the filtering test.

All those machines which are registered themselves with the Grid and are running Grid OS module in their kernel must have some static information attached with them. Such as; CPU processing speed, RAM capacity, LAN connectivity.

This is the information on the basis of which resource filtering will be performed. Broker has already extracted the job’s constraints information. It will pass on these constraints to all of the authenticated discovered machines inside the Grid. So each machine in the Grid receives those job constraints and only that machine will respond back to the Broker client which fulfils those minimum set of constraints. Hence those set of machines which do not fulfil the minimum requirement criteria are filtered out right from the beginning. Now only limited set of machines (resources) fulfilling the minimum application requirements is left in the system. The machines fulfilling the requirements respond back to the Broker client along with their dynamic resource descriptions in order to compete for the Job.

In this project just the above highlighted constraints for the job are considered i.e. CPU processing speed, memory and required library or DLL. For example if Job requirement for the minimum CPU processing speed is 2.6 GHz and minimum RAM capacity of 256 MB, then those machines which has CPU processing speed less than 2.6 GHz and RAM less than 256 MB will not respond to the Brokers request. And in other case if the application requires some specific libraries and DLLs at the remote location then all those resources will be filtered out from the competition if these required libraries and DLLs are not present there. An important consideration is when an application is developed using JAVA, because JAVA applications require specific JVM installed in the machine without which any JAVA application is unable to run. So this will also be the task of resource filter service of the broker to neglect all the machines without having proper version of JVM installed in it. And similarly C/C++ applications require glibc for the applications to execute successfully. Hence detailed investigation is performed only on that reduced set for selecting the most optimum machine among them. In other words it can be said that the DRD service will only run for that reduced set of machines.

This is very critical in order to reduce the congestion and traffic load across the network. It surely improves the efficiency of the system. Because only those machines which are eligible for job execution will send their dynamic resource descriptions across the network, and other do not even respond back.

6.3. System Selection

So finally Resource Broker client will get its input from DRD service in the form of dynamic description of resources. After getting this input the Broker will perform appropriate calculations on it in order to select the best optimum node for the job so that it will complete its execution there. The broker will basically act as a Match-Maker i.e. matching available resource to the user's request. Broker will select a machine after rating all of the machines. For this it will implement a rating algorithm to rank the machines.

There are few things to consider by the Broker which are; CPU utilization and CPU cycles availability, Memory availability, Bandwidth utilization, Network latency. As already described in this project, three factors are being considered i.e. CPU, memory and data rate available across the network (bandwidth).

Broker has to make calculations for Time and Cost factors and it will select the node with having the greatest response time (i.e. the greatest turnaround time for the submitted application) and the lowest incurring cost which will definitely depends upon the processor and memory availability along with the communication channel and network characteristics which are considered by the Broker while making its final decision regarding optimum system selection.

Main task of a resource broker is to perform matchmaking i.e. matching available resources to user's request. Matchmaking performed by broker is described in Figure 6.2 below.

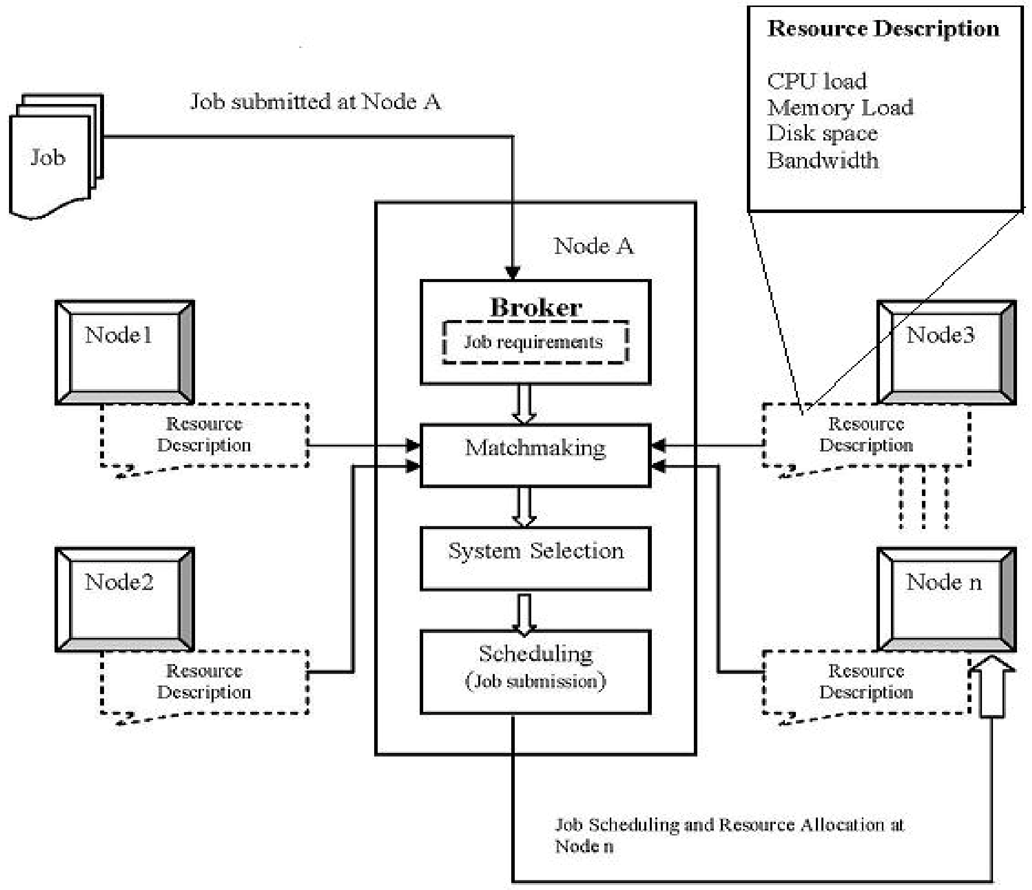


Figure 6.2: Architectural overview of the Matchmaking Framework

6.4. Performance Metrics for the Resource Broker

There must be some metrics and criteria on the basis of which broker will select the best optimum node. At the end what Broker wants is; The machine which gives maximum processing speed. This will depend on the processing speed of the available machine and also how much free CPU it has to entertain the job. This metrics is important for the computational intensive jobs, the machine which gives the maximum RAM storage capacity. This will be important for the applications having very large footprint. Efficiency is achieved only in the case when all of the instructions and data reside simultaneously inside the RAM, the machine which after being accessed provides minimum network latency. This is definitely dependent on the network traffic on the path and on the frequency at which other machines on the network are accessing the machine or at least accessing the same network path. This will be important for the communication intensive applications, also Broker has to take care of the data and files which are required by the application to complete its execution and produce results. The required data and files must be present at the remote machine. But for the time being this factor is not considered in this project until distributed data and file management system for Grid OS is completed, another important requirement is the presence of standard API libraries and includes files required by the executable of the application. Without which application will be unable to execute at the remote machine. This is one of the static constraints and all the machines not fulfilling this are filtered out right from the beginning and in the end, the required result with respect to user is the machine which provides minimum turnaround time for the Job or in other words the machine having the quickest response back to the client along with the results.

6.5. Summary

In this chapter the detail of the working of resource broker and its subcomponents is described.

Chapter 7

7. Heart Beat Monitoring

7.1. Introduction

This chapter introduces the concept of Heart Beat Monitoring and its inclusion in ARDIG. Heart Beat Monitoring is the process where a server monitors the availability of other nodes in the cluster. Heart Beat monitoring is used in two contexts in ARDIG, in monitoring node availability rate and process level node activity.

7.2. Monitoring in ARDIG

ARDIG heartbeat monitoring is completely decentralized except for the HB monitor which monitors node availability rate. This component is responsible for monitoring nodes [18]. It initiates a polling thread which checks each node availably and reports back to Super Node which updates the mean available time for each Node. It's another major task is to monitor the subGrid workload report to regional Peer incase subGrid goes below par. Figure 7.1 is the activity diagram of the heart beat monitoring in ARDIG and figure 7.2 shows the algorithm of how to determine where the subGrid is below the threshold level in ARDIG.

- Let threshold be '**th**' define the determining factor to go for optimizing or not. Its calculation mechanism is discussed in coming section.
- Let rating of each subGrid be define as '**subRatin[x]**' where x is the id of subGrid

Optimization process is initiated in case any '**x**' is below **th** factor is true.

subRating[x]<th

Fig 7.1 Algorithm to determine where subGrid is below threshold

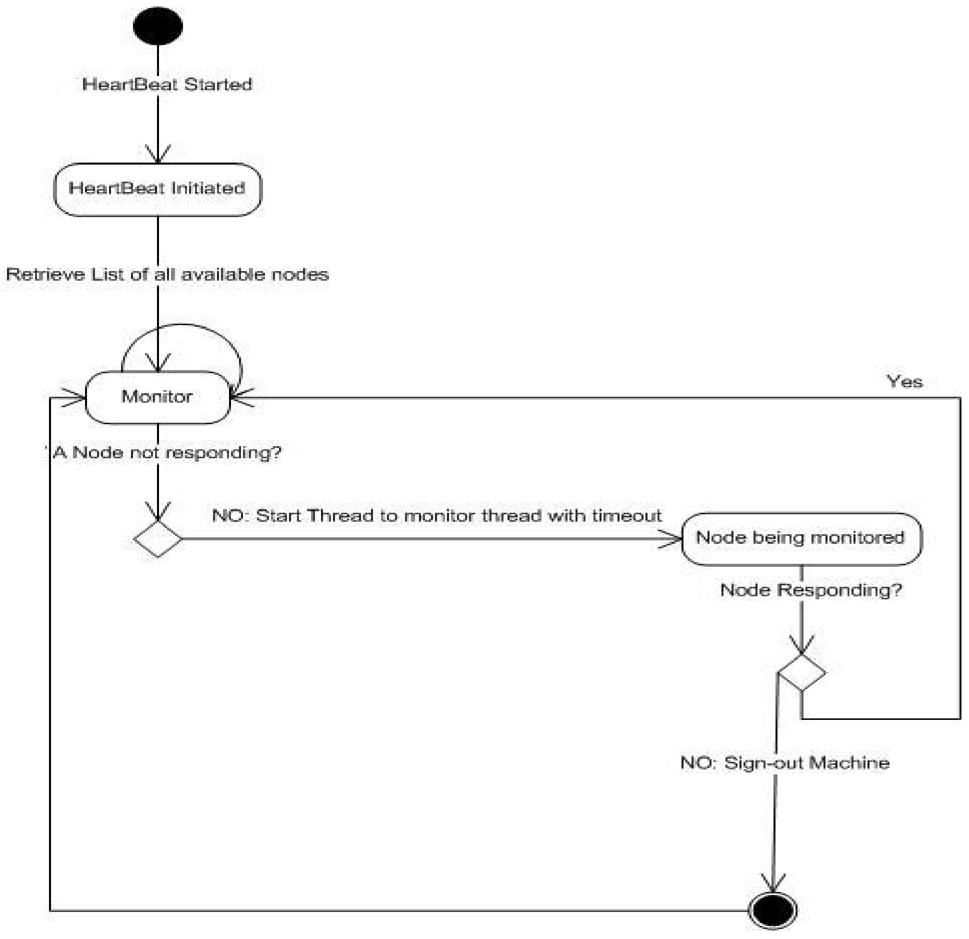


Fig 7.1 Hearts Beat Monitoring Activity Diagram

Heart Beat Monitoring is an important component in maintaining fault tolerance and stability in a ARDIG cluster and maintaining a good level of quality of service.

7.3. Summary

Heart Beat Monitoring is an important component in maintaining fault tolerance and stability in a cluster and maintaining a good level of quality of service. Two types of Heart Beat Systems are deployed Node Churn, which is used by the Super peer, and the process-level which is used by nodes which are executing processes remotely.

Chapter 8

8. Analysis and Testing

8.1. Introduction

Numerous algorithms have been developed for resource brokering in Grid environments [24]. Most are customized to Grids working at the virtual organization (VO) or cluster level granularity. Grid OS aims to support both interactive user applications and resource intensive Grid applications. For interactive user applications resource brokering with machine-level granularity is required, whereas Grid applications require cluster level granularity. Here discussion is restricted to grid enabling interactive user applications which are not supported in the existing Grid infrastructure. Resource brokering and scheduling algorithms which work at machine level granularity are mostly derived from intra-cluster level algorithms.

8.2. Self Healing Testing

Testing was done with three Grid Node and one super node and all the possible test cases were carried out to determine the effectiveness of the system.

8.3. Simulation parameters and analysis

Simulation was done with GridSim and the following are the simulation parameters. Grid environment was emulated with 30 subGrids each with 10 GridNodes and a total of 300 GridNodes.

Jobs were varied from 1 to 100 GridLets per subGrids. Job Specfication are as follows; Gridlet length describes the mips rating for the job was varied from 2000 to 3000., Gridlet file size that varies within the range 100 + (10% to 40%), Gridlet output size that varies within the range 250 + (10% to 50%).

The resource Characteristics for the GridNodes are as follows; Speed =*300 to 2000 Mips* Bandwidth= *100 to 1000 MB*, Memory= *128 to 2048 GB*, Availability= *(0,1)*, peak Load = 0.5, Off Peak Load = 0.2

The result followed by the graph shows the total processing cost in actual CPU time with respect No of GridLets. The graph shows three different scenarios, red one is the cost of processing without job migration, blue one is with migration and the last one in little green shows the cost after the subGrids have been organized.

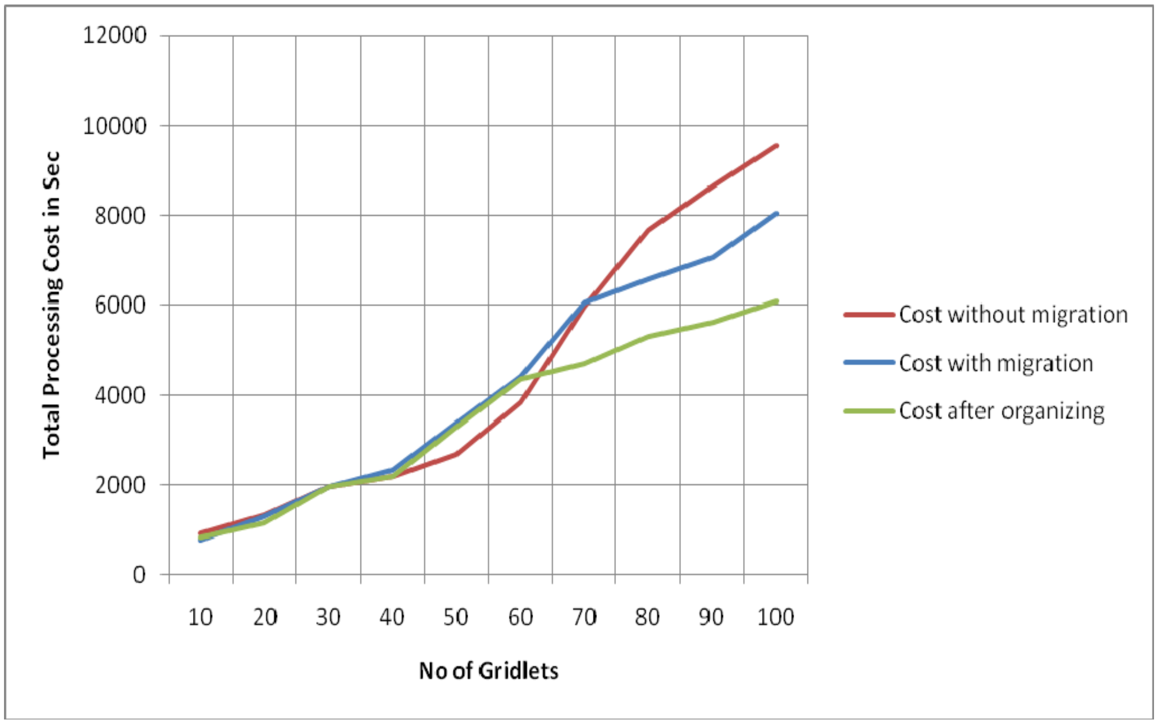
8.4. Simulation Result

The table below shows three different result set for three different scenarios. In each result set the no of GridLets were increased by 10 to check the corresponding effect on the total processing cost in seconds of the subGrids.

No of GridLets	**Total Cost with migration**	**No of GridLets**	**Total Cost without migration**	**No of GridLets**	**Total Cost after organizing**
10	789.0282299	10	909.9495616	10	834.6781595
20	1314.69789	20	1301.481755	20	1146.73238
30	1963.578062	30	2321.987233	30	1946.167598
40	2339.632028	40	2169.977062	40	2186.705823
50	3387.617063	50	2684.91103	50	3275.155291
60	4423.494961	60	3347.310068	60	4365.642813
70	6065.173859	70	5972.556734	70	4714.741766
80	6187.643585	80	7673.393012	80	5304.134499
90	5495.314802	90	5895.314802	90	4861.958735
100	9356.003681	100	9564.517624	100	6187.693023

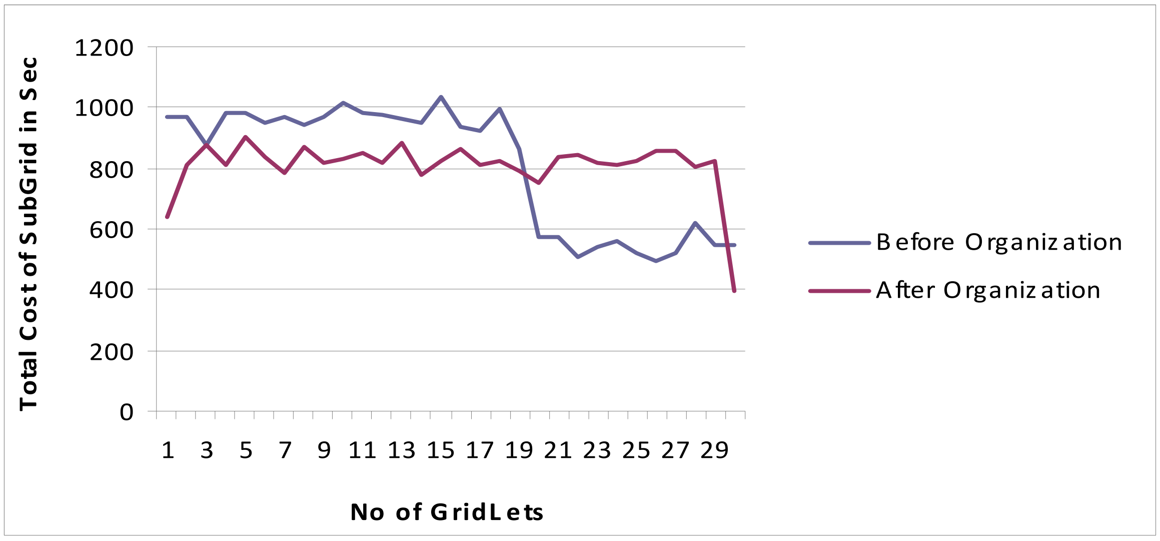
Table 8.1: Simulation Result of three difference scenarios

8.5. Simulation Graph



Graph 8.2: Performance of Self Organization algorithm

The graph below shows the cost before and after organization.



Graph 8.3: Performance of SubGrid after optimization

The first graph 8.2 shows the one with maximum cost and increase due to the fact that each job is taking more time at later stage resource is becoming busier. While the scenario with job migration has little less increase rate as in case of increase in machine load job is migrated to resource with more free resource. It is depicted from the result that the rate of increase in cost after the organization of nodes is much less as compare to the others. This is because the amount of job migration from one subGrid to the other has been reduced

8.6. Simulation Discovery Code

Refer to appendix-B

8.7. Summary

This chapter demonstrates show the discovery algorithm used in ARDIG is most efficient in a number of environments, hence proving its suitability to multiple user-centric computing environments.

Chapter 9

9. Epilogue

9.1. Conclusion

Grid computing has made promising progress, there are many projects involved in developing high performance Grids but the fact still remain that such huge projects need to have the autonomic infrastructure to deal with dynamic, pervasive and decentralize nature of Grid. At this stage, a Grid service can provide transient and state full services, and it also has some knowledge about itself. For example, what kind of input and output it semantically needs and what kind of domain it applies to. However, it does not have any autonomic features. It is envisioned that in future Grid system would compose of autonomic Grid Services.

Our proposed architecture would provide a solution to numerous issues, due to lacking autonomy in existing systems which required thorough analysis, extensive Grid Computing knowledge and time to deal with. It would automate the low level task buying more time to concentrate on other issues.

9.2. Future Work

In future this system would be integrated with Phantom OS to make Phantom OS decentralized, fault tolerant and pervasive. And also make the system compatible with DIANA schedulers. Apart from this another main work area is node access.

As nodes are assumed to have static and direct access in our project but in reality this would be real so solution to deal with issue involving accessing GridNodes behind Gateway and proxy would be required.

Regional node dependency needs to be eliminated as well. This can be done by make use of overlay network like PGrid.

Appendix- A

1 Self Organization Algorithm

```
private void selfOrganize()
{
      System.out.println("self organize");
      int i=0;
      int j=noSubGrids-1;
      rateAllSubGrids(A,B,C,D);
      double mean=calculateMean();
      System.out.println(mean);

      for (int i1=0;i1<noSubGrids;i1++)
            sortSubGrid(i1,A,B,C,D);

      int mid=noSubGrids/2;
      while (i<mid && j>mid)
      {
            shNode(j,i,mean,A,B,C,D);
            i++;
            j--;
      }

}
```

2 Node Shifting Algorithm

```
private void shNode(int subGridDId,int subGridTId,double
mean,double wS,double wM,double wB,double wA)
     {
          System.out.println("Shifting nodes");
          double
drating=rateSubGrid(subGrids[subGridDId],wS,wM,wB,wA);
          double
trating=rateSubGrid(subGrids[subGridTId],wS,wM,wB,wA);

          double dNrating=0;
          double tNrating=0;

          int i=0,j=subGrids[subGridDId].size()-1;
          int dId=0,tId=0;;
          while (drating>mean && trating<mean){

               dId = ( (Integer)subGrids[subGridDId].get(j)
).intValue();
               dNrating=nodeRating(dId,wS,wM,wB,wA);

               tId = ( (Integer)subGrids[subGridTId].get(i)
).intValue();
               tNrating=nodeRating(tId,wS,wM,wB,wA);

               if ((drating-dNrating)>mean)
               {
                    trating+=dNrating;
                    drating-=dNrating;

     subGrids[subGridTId].add(subGrids[subGridDId].get(j));
                    subGrids[subGridDId].remove(j);
                    j--;
               }else if ((trating-tNrating+dNrating)<mean)
               {
                    trating=trating-tNrating+dNrating;
                    drating=drating-dNrating+tNrating;
                    Object
temp=subGrids[subGridTId].remove(i);
                    subGrids[subGridDId].add(temp);

     subGrids[subGridTId].add(subGrids[subGridDId].get(j));
                    subGrids[subGridDId].remove(j);
                    i++;
                    j--;
                    }
               }

          }
```

3 Mean Calculations

```
          private double calculateMean()

          {

               System.out.println("calculate mean");

               double average=0;

               for (int i=0;i<agRating.length-1;i++)

                    average+=agRating[i];

               average=average/agRating.length;

               return average;

          }
```

4 Super Node Election Algorithm

```
     private int SuperNodeElection(LinkedList resList)

          {

               System.out.println("super node election");

               int tempid=0;

               double tempc=0;

               String tempn="";
               int resID[] = new int[resList.size()];
               double resCost[] = new double[resList.size()];
               String resName[] = new String[resList.size()];
               ResourceCharacteristics resChar = null;

               int i = 0;
               // a loop to get all the resources available
               for (i = 0; i < resList.size(); i++)
               {
                    // Resource list contains list of resource
     IDs not grid resource
                    // objects.
                    resID[i] = ( (Integer)resList.get(i)
     ).intValue();

                    // Requests to resource entity to send its
     characteristics
                    super.send(resID[i],
     GridSimTags.SCHEDULE_NOW,

          GridSimTags.RESOURCE_CHARACTERISTICS, this.ID_);

                    // waiting to get a resource characteristics
```

```
                resChar = (ResourceCharacteristics)
super.receiveEventObject();
                resName[i] = resChar.getResourceName();
                resCost[i] = resChar.getCostPerSec();
                if (resChar.getMIPSRating() > tempc)
                {
                     tempid = resID[i];
                     tempc=resChar.getMIPSRating();
                     tempn=resName[i];
                }

           }
           System.out.println();
           System.out.println("ELECTED SUPER PEER IS ### ID:
" +tempid +" # NAME: " + tempn + " # COST: " + tempc  );
           System.out.println();

           return tempid ;
      }
```

Bibliography

[1] Globus: A Metacomputing Infrastructure Toolkit. I. Foster, C. Kesselman. Intl J. Supercomputer, Applications, 11(2):115-128, 1997.

[2] D. W. Erwin & D. F. Snelling, UNICORE: A Grid Computing Environment, Lecture Notes in Computer Science, Springer 2001, Volume 2150, pages 825-834.

[3] Global Grid Forum, Global Grid Forum Conference, March 4-9, 2001, Amsterdam.

[4] P Padala and J N Wilson. GridOS: Operating System Services for Grid Architectures. In Proceedings of International Conference On H.P computing 2002

[5] IntelItanium2,http://www.intel.com/products/server/processors/server/ itanium2/.

[6] C. Mastroianni, D. Talia & O. Verta, A Super-Peer Model for Building Resource Discovery Services in Grids: Design and Simulation Analysis, EGC 2005, LNCS, Volume 3470, pages. 132-143 .

[7] R. Raman, M. Livny & M. Solomon, Resource Management through Multilateral Matchmaking, Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, Pittsburgh, Pennsylvania, August 2000, pages 290-291.

[8] P. Trunfio et al., Peer-to-Peer Models for Resource Discovery on Grids. In Proc. of the 2nd CoreGRID Workshop on Grid and Peer to Peer Systems Architecture, Paris, France, January 2006

[9] B. Yang, H. Garcia-Molina, Designing a Super-peer Network, In Proceedings of the 19th International Conference on Data Engineering (ICDE), March 2003, Bangalore, India.

[10] Jeanvoine, E.; Rilling, L.;Morin, C; Leprince, D, "Using Overlay Networks to Build Operating System Services for Large Scale Grids**",** The Fifth International Symposium on Parallel and Distributed Computing, 2006. ISPDC '06, July 2006

[11] C. Mastroianni, D. Talia & O. Verta, A Super-Peer Model for Building Resource Discovery Services in Grids: Design and Simulation Analysis, EGC 2005, LNCS, Volume 3470, pages. 132-143 .

[12] R. Raman, M. Livny & M. Solomon, Resource Management through Multilateral Matchmaking, Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, Pittsburgh, Pennsylvania, August 2000, pages 290-291.

[13] P. Trunfio et al., Peer-to-Peer Models for Resource Discovery on Grids. In Proc. of the 2nd CoreGRID Workshop on Grid and Peer to Peer Systems Architecture, Paris, France, January 2006

[14] B. Yang, H. Garcia-Molina, Designing a Super-peer Network, In Proceedings of the 19th International Conference on Data Engineering (ICDE), March 2003, Bangalore, India.

[15] C. Yang et al., Resource Broker for Computing Nodes Selection in Grid Computing Environments, GCC 2004, LNCS Volume 3251/2004, ISSN 0302-9743, pages 931-934

[16] Jeanvoine, E.; Rilling, L.;Morin, C; Leprince, D, "Using Overlay Networks to Build Operating System Services for Large Scale Grids**",** The Fifth International Symposium on Parallel and Distributed Computing, 2006. ISPDC '06, July 2006 Page(s):191 – 198