# PERVASIVE SELF HEALING AND NETWORK MANAGED OPERATING SYSTEM


By

NC Irfan Habib (Leader)

NC Kamran Soomro

PC Qasim Abbas

GC Habib-ur-Rahman

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

April 2007

**Abstract**

*Grid computing has progressed during the last decade due in part to the adoption of standardized Grid middleware such as Globus, gLite and Legion. However the pervasive adoption of Grid computing has been hampered by obstacles in user-centric computing environments, most notably the scant support of the Grid middleware for interactive applications, inability to autonomously self-reorganize to accommodate scale and its apparent lack of plug and play capability, aswell as the steep learning curve associated with setting up Grids with existing Grid Middleware. In this project we aim to develop a Grid Operating System (Grid OS), PhantomOS, and implement components of its architecture and show how it removes many of these technical barriers. The pervasive applicability of a Grid OS will be demonstrated with potential scenarios that would become realizable.*

# 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.

# Dedication

We would like to dedicate this project to our parents for their unwavering support and to the talented, energetic faculty of our college for their guidance and effort.

# 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, **Maj 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 **DG NIIT Dr. Arshad Ali** who supervised and managed all the research activities related to the project. Also, we are grateful to our foreign supervisor, **Dr Richard McClatchey (CERN),** who enabled our work to meet the required standards and get exposure internationally. And we must say thanks to Dr Aashiq Anjum for his devoted guidance and support.

## LIST OF PUBLICATIONS

1. **"From Grid Middleware to Grid Operating System"**
   Published in the *5th IEEE International Conference on Grid and Cooperative Computing*, Oct 22-24, 2006, Changsha, China
   Presented by: Dr. Richard McClatchey, CERN/UWE
   Available at: IEEE Digital Library

2. **"Grid Operating System: Towards a Pervasive Grid Computing Platform"**
   Submitted to *Elsevier Future Generation Computer Systems: International Journal of Grid Computing*

3. **"Getting Started with Condor"**
   Published on *ACM Linux Journal, September 2006*
   Available at: ACM Digital Library
   http://portal.acm.org/ft_gateway.cfm?id=1152901&type=html&coll=ACM&dl=ACM&CFID=16151968&CFTOKEN=26126555

4. **"Xen"**
   Published on *ACM Linux Journal, May 2006*
   Available at: ACM Digital Library
   http://portal.acm.org/ft_gateway.cfm?id=1134164&type=html&coll=ACM&dl=ACM&CFID=16151968&CFTOKEN=26126555

5. **"Getting Started with LIDS"**
   Published on *ACM Linux Journal, March 2006*
   Available at: ACM Digital Library
   http://portal.acm.org/ft_gateway.cfm?id=1119450&type=html&coll=ACM&dl=ACM&CFID=16151968&CFTOKEN=26126555

6. **"Integrating Perl and PHP"**
   Published on *ACM Linux Journal, February 2007*
   Available at: ACM Digital Library

7. **"Ltools"**
   Published on *ACM Linux Journal, February 2007*
   Available at: ACM Digital Library

8. **"Creating SELinux Policies Simplified"**
   Published on *ACM Linux Journal, February 2007*
   Available at: ACM Digital Library

9. **"State of the Art In Grid Computing",**
   Accepted at the IEEE International Symposium on High Performance Distributed Computing, Monterry Bay, California

# Table of Contents

# List of Figures

# List of Tables



# List of Graphs

Chapter 1

# 1. Introduction

## 1.1 Introduction and Background

Despite having made substantial advances during the last decade Grid computing is still neither pervasive nor widely deployed. In 2003 Gartner [18] predicted that in 2006 Grid computing would mature enough to leave the science laboratories and make its entry into the business world. So far there are only a few success stories [8], since only a subset of business applications are supported by existing Grid infrastructures. To date the computing research community and particularly eScience projects have been the biggest beneficiaries of Grid computing whereas other communities, such as the common home user and small-scale businesses, do not have access to a Grid customized to their needs nor have the capability to easily establish a working Grid. This gap in adoption can be traced to some technical hurdles which arise as a consequence of the current approaches to Grid computing. We are of the opinion that these hurdles originate from the current middleware approach to Grid computing, as detailed below.

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 to solve mainly scientific compute and data intensive problems. The role of the Grid middleware in this paradigm was to 'glue' the clusters together to achieve interoperability. Notable Grid middleware include Globus [15], gLite [25] and UNICORE [13]. This approach has however raised some barriers to Grid adoption for

other fields, since the cluster-oriented Grids of today are not suitable for user-centric computation partly due to their complex operation and maintenance requirements. The main barriers [1] to the adoption of Grid computing that result from the strong focus of current Grid computing research on eScience are the support for limited application types (which mostly comprise highly parallel and batch applications), the potentially inflexible network topologies and the steep learning curve for configuring and maintaining a Grid with Grid middleware. All these limitations make Grid computing in its present incarnation unsuitable for the common user with little computing expertise. For example, a domain which is generally common-user centric is the biomedical field. While on the one hand biomedical research has witnessed tremendous growth in terms of adoption of technologies to facilitate biomedical research, on the other there has also been an exponential growth of data that is generated and needs to be assimilated and consumed by individual clinicians. To enable knowledge discovery and foster enhanced collaboration medical sciences have increasingly turned towards Grid computing [5,17,21]. One other area which has already adopted Grid computing is High Energy Physics Research (HEP). Computing environments radically differ in both cases, HEP Grids, which involve mostly non-interactive compute intensive batch applications which are fully supported in the existing Grid infrastructures, have to cater for limited privacy issues and the interaction of the HEP physicist is minimal with the Grid middleware. In contrast biomedicine, where data is governed by national/international regulations, mostly involves interactive data-intensive applications, and the interaction of the researchers in the field with the Grid middleware is more extensive. These kinds of applications are not easily supported in the existing middleware. The differences in the computing environments highlight the

need for a generic Grid computing system that is not specialized to a community of users, as is the case nowadays. The authors of this papers are of the opinion that the Grid Operating System is an important step towards such a pervasive Grid computing system.

In this project we propose an approach which aims at bridging the gap between user-centric computing and eScience-centric Grid computing, via a so-called Grid Operating System. In Figure 1, we outline our objective of the integration of the Grid functionality that is associated with middleware, in the machine operating system along with the execution environment provided by a modern cluster middleware, in order to make a single unified system: the Grid Operating System.

There are many interpretations of the word "Grid Operating System". In [1] the authors defined it to be "an operating system which transparently enables a user to peruse discovered distributed resources, to share resources in a decentralized fashion, to seamlessly launch and to migrate tasks on global resources giving the user an impression that (s)he is using the local resources and to enable the control and monitoring of executed processes on a global scale through local means". According to this definition few systems qualify as "Grid Operating Systems", the closest ones being distributed operating systems. We restrict our discussion to those systems which are under active development or are in significant use such as OpenMOSIX/MOSIX [31,4] and research/commercial projects such as XenoServers [35] and Apple xGrid [3]. We compare these projects against the goals we wish to achieve, and identify their shortcomings, which we set out to address in Grid OS.

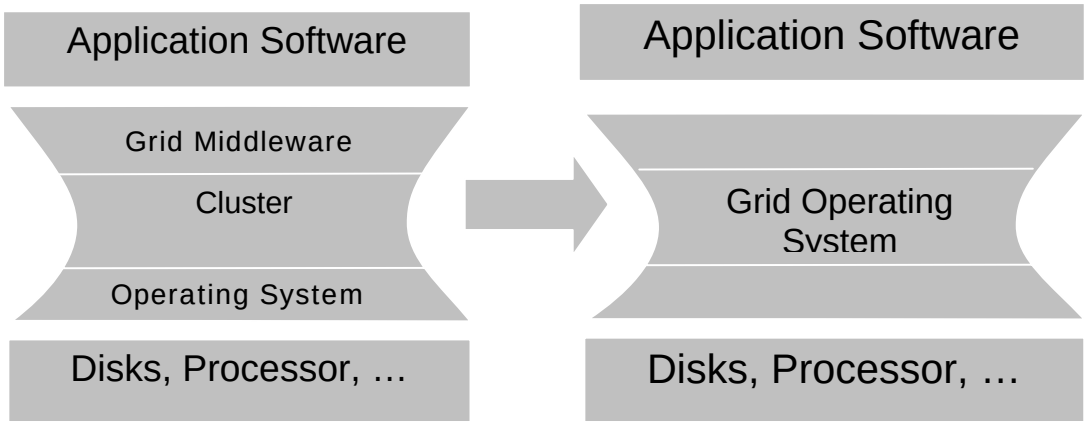



**Figure 1.1: From a Grid middleware based approach to a Grid OS approach**.

## 1.2 Problem Statement

To develop a user-centric pervasive Grid computing platform, which targets most of the technical barriers to adoption setup by the Grid Middleware

## 1.3 Scope

1. To implement a generic process migration mechanism, to eliminate the barriers to creating Grid Applications, and making it convenient to grid-enable interactive applications
2. To implement a decentralized resource broker and scheduling algorithm, to cater for scalability and robust architecture
3. To implement distributed Heart Beat monitoring which eventually should lead to a self-healing Grid
4. To implement process level fault tolerance to protect execution states

## 1.4 Related Work

This section compares and contrasts the work in the domain against our stated objectives for the Grid OS. The XenoServers project [35] aims at developing a network of globally distributed servers in which users can deploy any kind of untrusted and unverified computation. As it is targeted at the public, Xenoserver allows untrusted user computations. However, it does not as yet support the execution of all types of computations, such as multithreaded interactive desktop applications. Xen, a virtual machine monitor, which is a central component in XenoServers, allows multiple users to run applications on XenoServers, in a manner which is secure for both resource providers and owners and does not degrade the quality of service (QoS) of the system for both parties. Using virtualization to ensure QoS and security of the system is an essential feature for any distributed operating system. However Xen's resource requirements limit its utility for a vast majority of the desktop systems.

Apple xGrid [3] is a part of the Apple MacOS-X operating system, which enables an organization to create a Grid/cluster and to run computations on it. Apple xGrid is perhaps one of the first common-user oriented Grid computing systems. Jobs submitted by a user to an Apple xGrid system are divided into independent tasks by the 'Controller', a machine setup to coordinate the computations on the Grid. The tasks are dispatched to 'Agents' which are dedicated machines offering their computing resources for the execution of tasks. Apple xGrid has some drawbacks in that the division of the Grid into Agents, Controllers, and submission machines is unscalable due to the client

server nature of its interaction. The Controller in Apple xGrid may serve hundreds of users, but once the user-base increases to thousands or tens of thousands, the QoS drops radically. Furthermore, xGrid has not been deployed for large numbers of machines in multiple domains which can give a true indication of its scalability. Apple xGrid is not self-organizing, which might be the single most important hurdle to its transition towards a universal Grid testbed.

OpenMOSIX [31] is a distributed cluster operating system which traces its lineage from MOSIX [4]. It provides automated load balancing through a completely transparent mechanism of process migration and communication. Process migration and communication is an essential component in Grid-enabling interactive applications. However, OpenMOSIX has a number of drawbacks which include a multicast-based discovery service, which is not very robust and scalable for a widely distributed system. A feature that is conspicuously missing from OpenMOSIX is a resource-broker, as OpenMOSIX implicitly assumes a homogenous computing environment, and thus bases its decisions only on idle CPU times. OpenMOSIX lags in fault tolerance, hence limiting the utility of OpenMOSIX in a dynamic Grid environment. Important lessons can therefore be drawn from existing Grid systems implementations:

1) the Grid enabling mechanism must not be an overhead to the system, so as to severely limit the capabilities of the machine;
2) the Grid must not be managed from one central authority, otherwise scalability problems may arise, and
3) the system must cater for a dynamic Grid, consisting of heterogeneous resources.

Recently two major efforts in the direction of Grid Operating System have been launched: Vigne Grid Operating System[41], is a Grid OS which targets relieving users and programmers from the burden of dealing with the highly distributed and volatile resources of computational grids. Vigne focuses on three issues: Grid level single system image to provide abstractions for users and programmers to hide physical distribution of grid resources, self-healing services to tolerate failure and reconfigurations in the Grid and self-organization to relieve administrators from manually configuring and maintaining Vigne OS's services. Vigne is dependant on the Kerrighed Cluster system which manages issues like process/thread migrations, checkpointing, shared memory and distributed file systems. However Kerrighed has some limitations which would limit wide scale deployment, Vigne deals with self healing behavior of sites, where as in an individual site, Kerrighed does not tolerate node failures, Kerrighed clusters can not be bigger than 32 nodes and provide no SMP and 64 bit architecture support.

XtreemOS aims at the design and implementation of an open source Grid operating system with native support for virtual organizations(VO) which would be capable of running on a wide range of underlying platforms, from clusters to mobiles. XtreemOS plans to implement a set of system services to extend those found in a typical Linux system. These services will provide Grid computing capabilities to individual nodes. The aims of XtreemOS are similar to the PhantomOS project, both are Linux based and open source and try to develop an OS level Grid solution with support for grid enabling

application and providing self healing services for large scale dynamic Grids. XtreemOS focuses on developing a solution from clusters to small scale mobile devices.

## 1.5 Organization of Thesis

Chapter 1, deals with introducing the problem statement and the related field of work.

Chapter 2, deals with the overall architecture of PhantomOS and how components relate to each other

Chapter 3 to 8 Starts the series of chapters which methodically deal individual components of PhantomOS

## 1.6 Summary

This chapter has emphasis on the introduction of our area of interest, importance and motivation for the project, aim and background for selecting this project and then the problem statement based on which project objectives are finalized.

Chapter 2

# 2. Architecture

## 2.1 Architecture of PhantomOS

Contemporary Grids fall into two models: the adhoc Grid model and the cluster-based Grid model. The adhoc Grid model involves the creation of servers which coordinate the activities of the Grid and execution machines, which execute the tasks for the servers. Most desktop oriented Grid projects such as BOINC [2] and Entropia [7] use this model. Its advantages are that Grids can become arbitrarily large and the combined resources can be pooled together for distributed computation. This approach however has some disadvantages for users that are part of the Grid. The users provide their resources for computation of foreign jobs and they cannot use the same resources for their applications. Interactivity is very restricted between the nodes leading to poor QoS and there are high latencies in this form of computing which is further aggravated due to very limited control over the nodes' resources. Moreover, this model is only useful for some compute applications and little progress has been made for the data Grid, since the network and storage capacities are not considered as resources in such Grids. Such Grids also rely on users' goodwill to exist since they mostly employ opportunistic resources and thus they do not provide a sustainable business model. Additionally fault tolerance is low in these systems and scheduling decisions are made with a best effort strategy.

In the cluster-based Grid model, the power of clusters located around the world is combined. The most powerful of the contemporary Grids follow this model [32]. These

Grids are operated in centralized environments, and often have dedicated resources connected with high-speed network links. The Grid relies on certain servers which centralize important functionality such as resource brokering, scheduling etc.; the constituent clusters are glued together via a Grid middleware. Moreover, due to central management, if any central server fails, large parts of the Grid can also fail. To execute jobs on the cluster, Grid middleware relies on cluster level execution services such as Condor [27], PBS [6] etc. Cluster level execution services have some limitations [11], which limit their use in multithreaded applications and in some circumstances require the modification of the source code to use certain features, such as checkpointing in Condor. Thus these limitations severely reduce the types of applications which can be executed on clusters. Most of the jobs that run on the cluster-based Grids are non-preemptive due to latency issues. However, in contrast to the adhoc Grids these are generic and allow any member of the Grid to execute any supported job on it.

Our Grid OS project aims at the convergence of common user and business oriented computing with eScience-centric Grid computing. The system should peruse distributed resources with minimal configuration, and transparently Grid-enable desktop applications to provide enhanced QoS to users. Grid OS aims to achieve this by drawing from a convergence of computing research fields such as virtualization, peer-to-peer (P2P) networking, Grid and cluster computing and operating systems. To develop the Grid OS system, which both aims at maintaining the scalability and user-oriented view of adhoc Grids as well as maintaining the scale of resource sharing in cluster-based Grids, we propose the development of a hybrid model which merges both adhoc and cluster-

oriented models. The topology we have adopted aims to enable the creation of both cluster oriented grids, where individual clusters represent the virtualized combined resources of disparate machines, and ad-hoc grids to enable groups of users to form virtual organizations in order to share their resources.

Introducing Grid computing features to the desktop presents challenges in terms of catering for the unique needs of desktop applications, maintaining autonomy, scalability and security of the system. Other challenges include making the system generic and sufficiently non-intrusive so that lay users can take advantage of the Grid computing features of the system and adopt it to different environments. Security is another area heavy with challenges. Businesses and users will find it an unacceptable risk using a system which allows them to offer their resources to foreign users but does not guarantee adequate security to their data. The authors of this paper are of the opinion that virtualization offers a cogent solution in ensuring security for both resource users and owners. On the other hand emulating the existing Grid infrastructures to run current grid applications in a generic system has its own set of challenges. Virtualization offers protection from untrusted applications, however existing Grids do not support untrusted applications. Rather they rely on certificates for identification of trusted computations. The Grid OS must support these mechanisms if it is to be viable for existing Grid environments. Other research issues include the virtualization of resources of simple machines into more powerful Grid nodes, to enable the execution of applications far beyond the reach of a single machine. For the scope of this paper we will restrict our

discussion of the Grid OS to grid-enabling the desktop for supporting common-user and business centric Grid computing.

Phantom OS has four essential components, which this paper addresses, which are designed to meet project aims and address the limitations of the existing Grid middleware:

- A transparent mechanism of 'grid-enabling' desktop applications, which includes distributed resource brokering
- A two-tier super peer [39] based mechanism to allow discovery of Grid OS nodes
- Fault tolerance in Grid OS
- Using virtualization to achieve security for resource owners and providers.

The above components are relevant to enabling the adoption of Grid computing in user-centric fields, components which are targeted of relieving existing Grid users of some of the drawback which arise from existing approaches to Grid computing, such OS level virtualization of heterogeneous resources to enable fine-grained resource management, allowing a wider array of application types to be run on Grids and simplifying grid application development will discussed in future publications.

| *Functions* | ***Grid Operating system*** | ***Grid Middleware*** |
|---|---|---|
| **Resource Management** | Fine grained kernel level resource management | Resource management based on prioritization of processes with no control over underlying hardware |
| **Support for Application** | Interactive/batch application, by using process migrations | Batch application only, involving scheduling of independent programs |
| **Resource Discovery** | Super peer to peer based to ensure scalability and QoS | Mostly client server |
| **Setup and configuration** | Involves installing utilities to control grid computing features of the operating system | Involves installation of multiple layers of software, e.g. cluster middleware, grid middleware, portals |

Table 2.1: Contrasting the Grid operating system with the Grid middleware approach

Table 2.1 contrasts the Grid OS with existing Grid middleware, making a case for why Grid OS will make Grids more relevant and pervasive in fields besides eScience.

## 2.2 Implementation

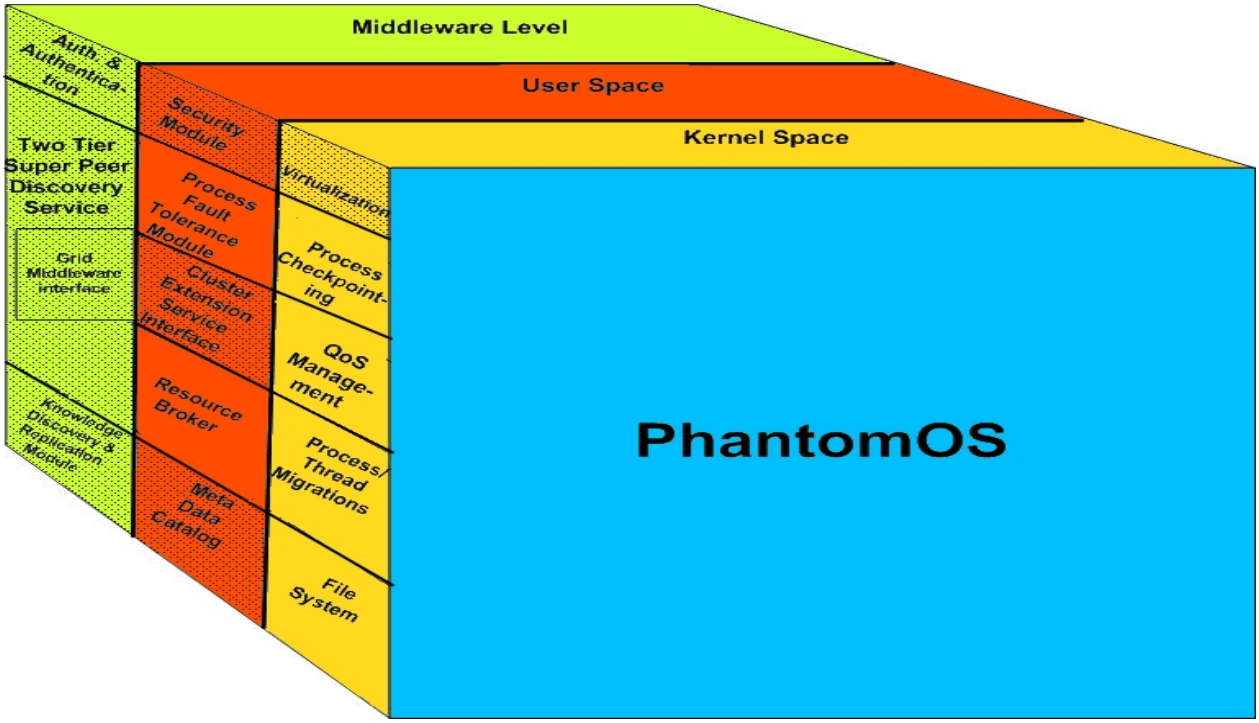



**Fig 2.1 PhantomOS Architecture**

PhantomOS as argued 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. Figure 1 shows PhantomOS from both perspectives, 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 they 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:

1. Process Checkpointing
2. QoS Management
3. Process Migrations
4. Resource Broker
5. Process Level Fault Tolerance (Includes HeartBeat Monitoring)
6. Parts of the Discovery Service

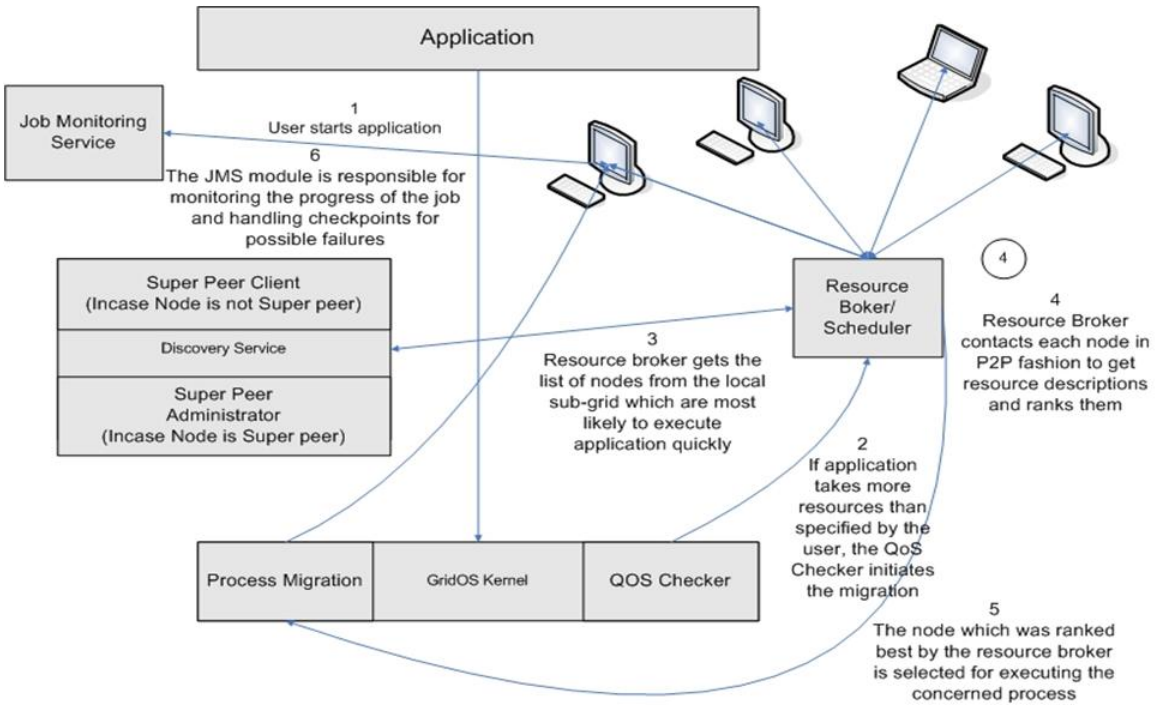



**Figure 2.2 PhantomOS Internals**

A General workflow is described. The above diagram displays the general interaction and flow of data between the components.

## 2.3 Summary

This chapter gives us the understanding about the architecture of PhantomOS and what components of PhantomOS which were implemented as part of the Degree Project. The following chapters will describe each component in more detail.

Chapter 3

# 3. Decentralized Resource Broker

## 3.1 Introduction

In recent decade there is an emergence and widespread adoption of Grid computing in different fields of the computer industry. There is a continued growth in both the application requirements and the complexity of the technologies used to meet those requirements. Grid computing is actually the sharing of heterogeneous and distributed resources to make full use of the underutilized resources in a collaborative environment. In today's computing world there are a large number of applications which requires huge amount of CPU processing and they consume large amount of CPU time. In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5 percent of the time. In some organizations, even the server machines can often be relatively idle. We will present a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage. Currently all the Grid computing solutions and implementations of Resource Broker are provided at the middleware and application level which does not provides the required efficiency and transparency to the end user, who is just concerned with the fast, efficient and responsive execution of its submitted task. Many Brokers who begun to address the needs of a true Grid level Broker, do not currently supports the full range of actions required in the brokering process.

So in order to utilize the maximum power of the Grid and exploiting underutilized resources inside the Grid there is an extreme need of a generic Broker which must be able to provide the required efficiency and transparency to the user in a generalized and dynamic way. The basic aim behind this proposed Resource Broker architecture is that now facilities for integrating computers in Grids should move from the middleware layer (toolkits) to the operating system layer, because an operating system is a more appropriate environment for providing Grid users access to resource sharing facilities. A Resource Brokering framework inside kernel must be designed to provide a virtual machine interface layered over the distributed, heterogeneous, autonomous, and dynamically available resources that compose a Grid. Approach is to integrate the Grid virtual machine as kernel Grid OS into Linux (turning Linux to Grid-enabled Linux). Broker inside GRID operating system must provide services such as providing simple connection to the GRID, handle and disseminate knowledge about the GRID resources in a peer-to-peer environment, offer access to GRID resources, identify the available and appropriate resources to be utilized within the Grid.

### 3.2 Resource Broker

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 they 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 turn around 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 we envision a brokering and scheduling engine built upon the underlying OS kernel which takes entire pool of resources available across the Grid.

We propose 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. Thus enabling compute intensive and memory intensive applications to make best use of the Grid resources in order to achieve high throughput. In our 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 as shown in Figure 2.1 above. Integrating the Grid Resource Broker with the OS kernel is one of the first steps towards providing transparent access to Grid resources for the Users applications.

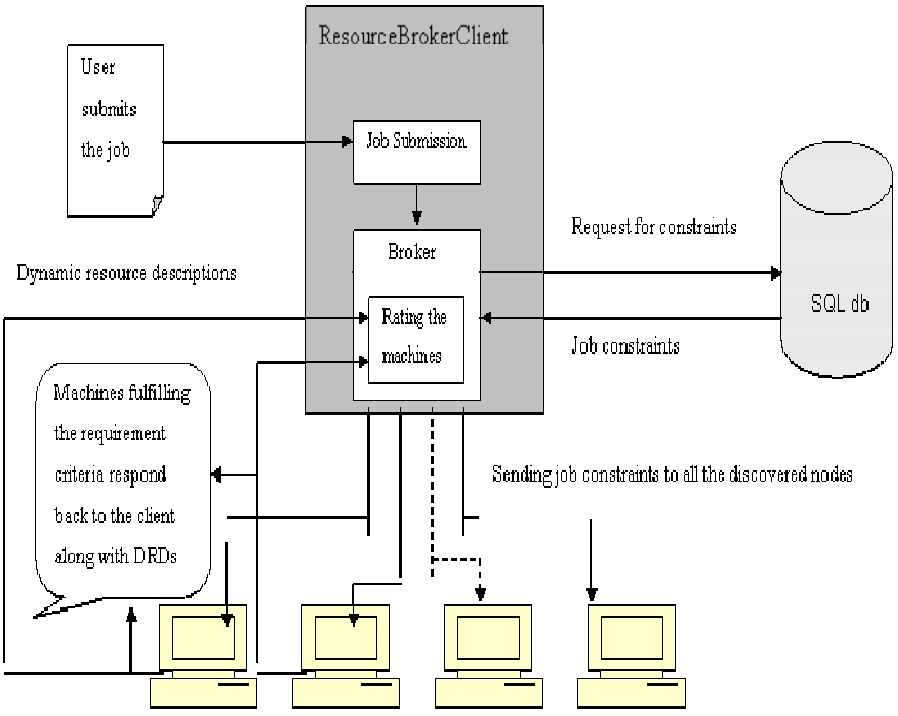



**Figure 3.1: Resource Broker architecture**

Figure 3.1 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. In this architecture the machine where the job is actually submitted by the user becomes resource broker client and all other machines temporarily act as resource broker server. Resource broker client and resource broker server are actually two separate services which run inside the brokering engine. Each machine inside the system can act both as a client and a server at the same time. The architecture and working of resource broker client and resource broker server and interaction among them is explained in the next section of this chapter. And implementation details of these services will be discussed in chapter 6 (Implementation of the Resource Broker).

### 3.3 Resource Broker Client and Server Interaction

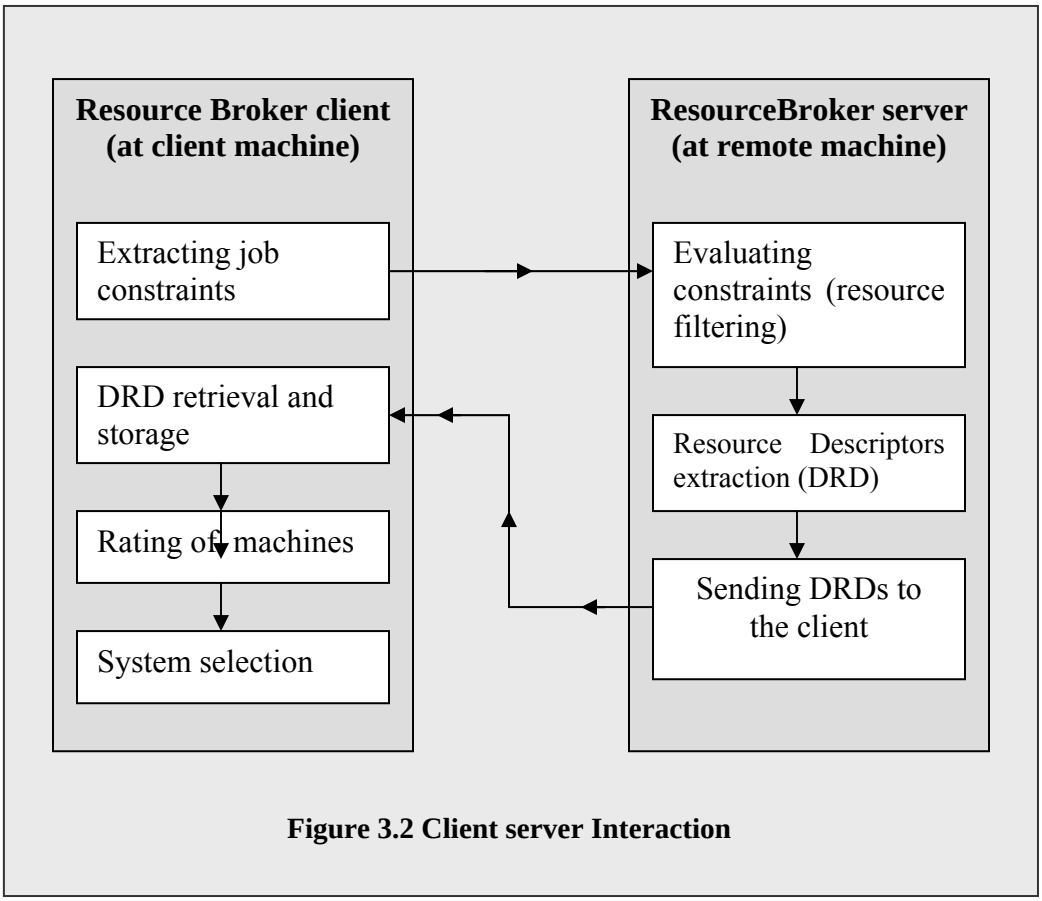



**Figure 3.2 Client server Interaction**

The proposed architecture in this project for the resource broker as shown in figure 3.2 involves resource broker client (resbclient) and resource broker server (resbserver) establishing a point-to-point communication link among each other across the network. The term client and server are used just to differentiate the machine where the job is submitted and the machine where the job is scheduled for remote execution. Otherwise in reality this is a true peer-to-peer Grid environment where all the machines in the system are acting as independent peers. Each machine in the system ran resbclient

and resbserver services. And each machine can both act as a client and a server at same or different instants.

### 3.4 Resource Broker Client (resbclient)

In this project user submits the job directly to the resource broker client. Then it is the task of the resource broker client to select the best machine across the Grid which provides maximum performance and turnaround time for the submitted job. The working of the resource broker client and its architecture is shown both in figure 3.1 and figure 3.2 above.

The actions performed by the resource broker client are:

- Receive the job submitted by the user
- Extract job constraints specific to this job
- Extract the list of authenticated discovered nodes across the Grid
- Sending job constraints along with its IP address (IP address of the client machine) as its identification to all of the discovered set of machines simultaneously. For this, resbclient creates separate thread for sending data to resbserver present at every remote machine.
- Receiving resource descriptors from the resbserver.
- Storing the resource descriptors locally and evaluating them to rate the machines after applying rating algorithm.

- Selected the machine with the highest rating value. Highest rating value is of that machine which provides greatest efficiency and optimum performance in terms of CPU processing, memory availability and network latency.

### 3.5 Resource Broker Server (resbserver)

Resource broker client service communicates with the resource broker server service running at each of the discovered machines . This service always remains in continuous listening mode, waiting to receive requests from resbclient. Only those machines will respond back to resbclient which passes the filtering test after evaluating job constraints. Each resbserver will evaluates itself. Resource broker server architecture and working is shown below in Figure 3.3

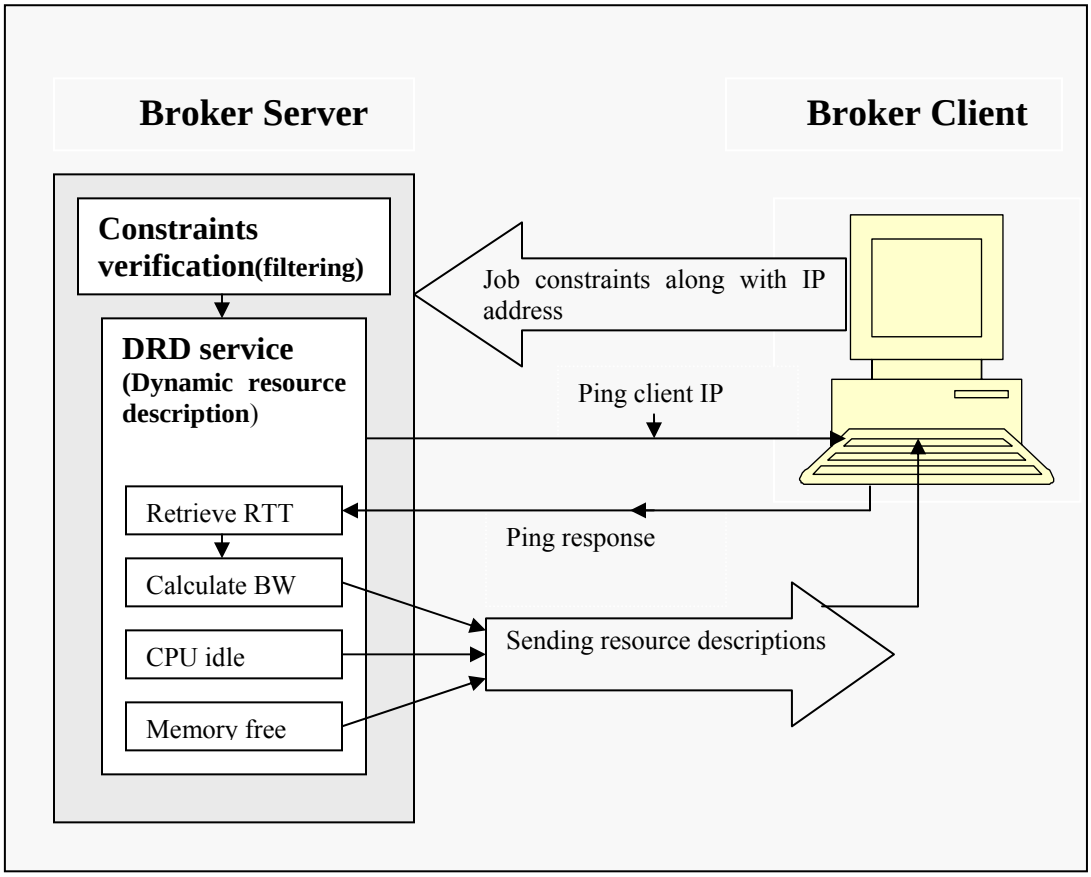



**Figure 3.3 Architecture of Resource Broker Server**

The resbserver is responsible for the following actions and tasks:

- Continuously listen for the request from resbclient.
- Receive job constraints and IP address of the machine where resbclient is running.
- Perform resource filtering through evaluating the job constraints by matching them with machine's native specifications.
- If constraints do not matches then resbserver simply filters out itself from the competition.
- If constrains are matched successfully then the resbserver extracts dynamic resource descriptions of the machine and send them to resbclient. This is performed by the DRD (dynamic resource description service) running at resbserver. The three dynamic descriptors extracted are CPU idleness, free memory and bandwidth. Bandwidth is the data rate available in bytes/second in accessing the machine. This bandwidth is calculated after retrieving the RTT for the client machine (see figure 3.3)

### 3.6 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 fulfills 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 GridOS 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 fulfills those minimum set of constraints. Hence those set of machines which do not fulfill the minimum requirement criteria are filtered out right from the beginning. Now only limited set of machines (resources) fulfilling the minimum application requirements are left in they 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 we just consider the above highlighted constraints for the job 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 then 2.6 GHz and RAM less then 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 we can say 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. Also there is fewer amounts of data to be stored and computed by the resbclient in making its decision for the optimum site selection.

### 3.7 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.

- CPU utilization and CPU cycles availability.
- Memory availability.
- Bandwidth utilization.
- Network latency.
- As already described in this project we are considering three factors 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 turn around 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 3.4 below.

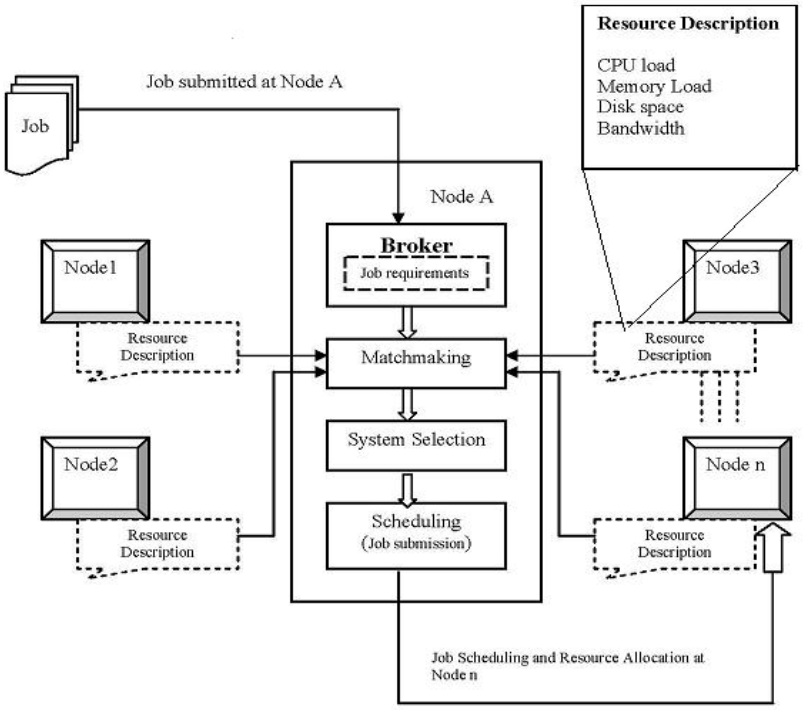



**Figure 3.4 : Architectural overview of the Matchmaking Framework**

## 3.8 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 turn around time for the Job or in other words the machine having the quickest response back to the client along with the results.

### 3.9 Dynamic Resource Description (DRD)

This service is running at each resource broker server. A single machine with the most optimized performance has to be selected from the available machines in the Grid upon which the Job will be scheduled. Hence in order to make the best possible Resource-Job

matching, detailed dynamic information about the resources must be needed by the Broker. The role of Dynamic Resource Description (DRD) service is to get the most updated status figures of the currently given set of selected machines (the filtered set of resources). Figure 3.5 below describes this process.

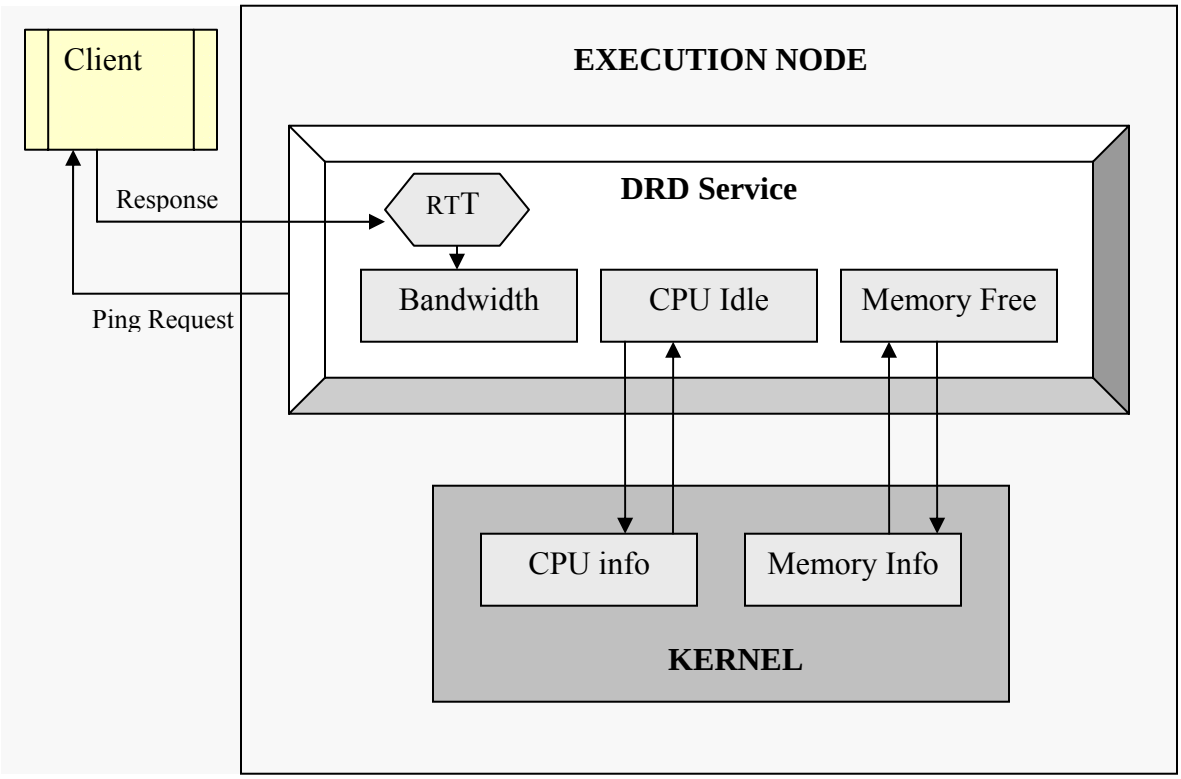



**Figure 3.5 Architecture of DRD service**

This service will retrieve the dynamic status figures, rather then knowing from the history. Dynamic resource descriptions are received by the resource broker client in response of the request which it sends to all of the discovered set of machines along with the Job constraints and its IP. These Dynamic Resource Descriptions will be

used by the Resource Broker in order to perform Match-Making among the submitted job and the available resources.

Some of the dynamic status figures as required by the Broker in order to perform Match-Making calculations are:

- **The current CPU utilization of each machine.**
- **The current available Memory status of each machine.**
- **Available bandwidth.**

In this project only the highlighted dynamic descriptions are considered. The working of DRD service is like condor ClassAdds based mechanism, where each node in the Grid will advertise its current status which is acquired by the Broker and finally use this information for Match-Making purpose.

## 3.10 Class Ads (Classified Advertisement) Based Mechanism

The ClassAds based mechanism for Matchmaking was first introduced by Condor High throughput computing (HTC), developed at university of Wisconsin. The Classified Advertisement (ClassAds) language facilitates the representation and participation of heterogeneous resources and users in the resource discovery and scheduling frameworks of highly dynamic distributed environments. Although developed in the context of the Condor system, the ClassAds language is an independent technology that has many applications, especially in the systems that exhibit uncertainty and dynamism inherent in large distributed systems. The Dynamic Resource Description (DRD) service in our proposed framework although not use Condor framework, rather uses the same methodology and mechanism to collect the dynamic information about resources from

distributed machines across the Grid. Every machine in the Grid will advertise its current status which is collected by the DRD service. And Broker will interact with the DRD service to retrieve all this information in order to make calculations and analysis for optimum node selection. As described above in section 3.9. this service is invoked by resbserver present at server-side resource brokering engine.

# Chapter 4

# 4. Process Migrations

## 4.1 Introduction

Process migration is the act of transferring a process between two machines. It enables dynamic load distribution, fault resilience, eased system administration, and data access locality. Despite these goals and ongoing research efforts, migration has not achieved widespread use. With the increasing deployment of distributed systems in general, and distributed operating systems in particular, process migration is again receiving more attention in both research and product development. As high-performance facilities shift from supercomputers to networks of workstations, and with the ever-increasing role of the World Wide Web, we expect migration to play a more important role and eventually to be widely adopted.

## 4.2 Process Migration In PhantomOS

In PhantomOS, process migration is used to transfer the processing load of the application to some other node that is better suited for the particular application. This is unique from other process migration mechanisms as it enables the migration of interactive applications. The user interface part of the application is kept at the current

node, while the processing is migrated to the remote node, thus reducing the execution time of the process greatly.

### 4.3 Process Migration Mechanism

When a process is migrated to a remote node, two processes are created. One is kept running at the home node and is called the Stub. The other is created at the remote node and is called the Deputy process. The Stub is responsible for maintaining the user interface, while the Deputy is responsible for handling the processing.

When processing is going on at the remote node in the Deputy process, the memory pages are constantly being modified. As soon as they are modified, they're copied to the Stub processes' address space. The Stub process now reflects the result of the computations that took place at the remote process. In this manner, the computation of the process is carried out at the remote node, while the user interface is kept at the home node.

The following figure shows how process migrations work:

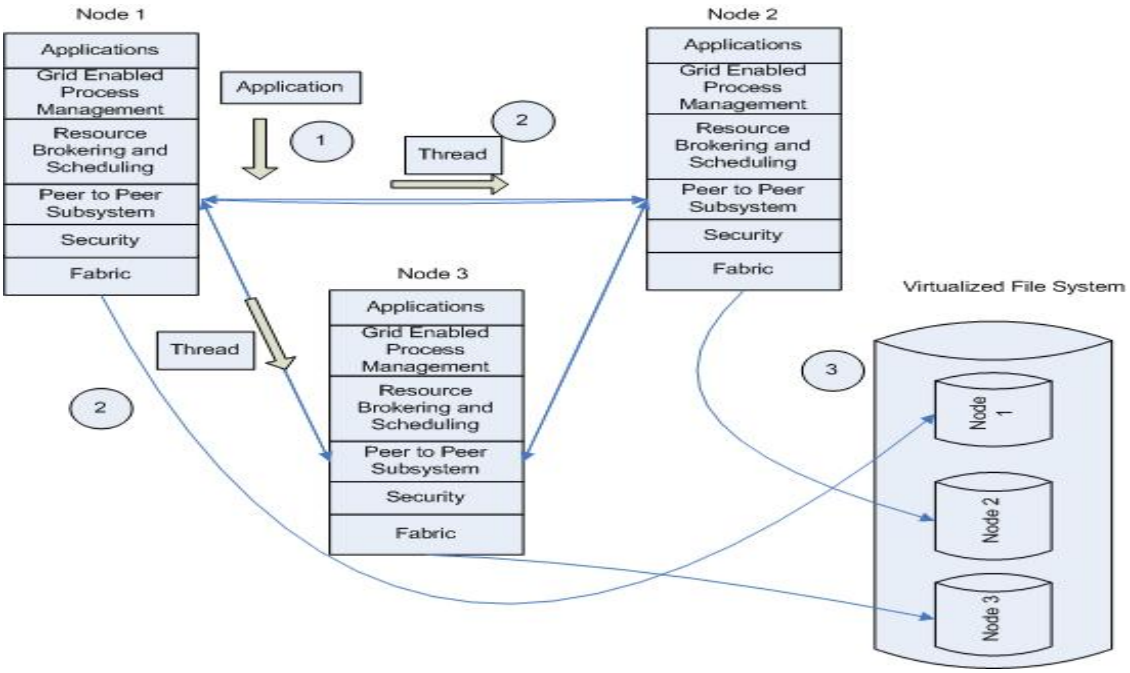



**Fig 4.1 Migration Architecture**

# Chapter 5

# 5. Process Level Fault-tolerance

## 5.1 Introduction

Existing Grid middleware have not been developed to be fault tolerant, rather they focus only on infrastructure. Without fault-tolerance, there would be no guarantee that any process that is migrated can ever be recovered once a failure occurs, additionally we cannot ensure, without the help of fault tolerance, the sustained operation of the Grid in a dynamic environment.

A node might crash, restart or a network link might simply go down, in which case even if migrated processes complete, there would be no means of communicating the results to the "home nodes" (we denote the nodes where the process originated as home nodes). In such a scenario, the system must be able to detect the failure, and recover from it. The process management fault tolerance method used in PhatomOS come from the cluster computing domain and is called checkpointing.

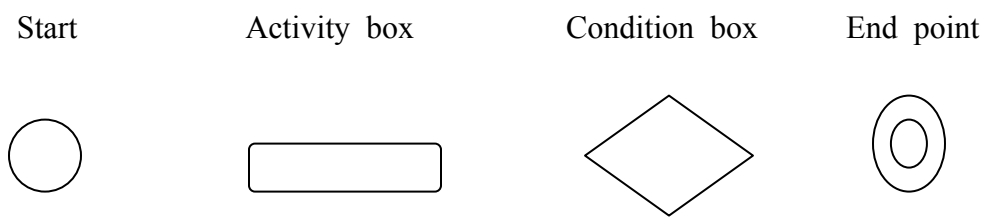
## 5.2 Checkpointing

Checkpointing involves saving the current state of the process to disk, and using it to resume the process should the need arise. Checkpointing is a crucial part of fault tolerance. However in order to support existing desktop applications, a checkpointing

mechanism is required which does not require the modification of the source code, hence checkpointing has to be handled at the OS kernel level. In PhantomOS architecture the job of the Job Monitoring Service (see Figure 3) is to track the progress of a job, and checkpoint it at regular intervals. The frequency of checkpointing, as it is pure overhead, is proportional to the instability of the machine. On a machine that has a history of many failures, the checkpoints will be made more frequently than on a machine with a more stable history. The checkpoints are creating at the home node of the process.

## 5.3 Chpoxd

To regularly checkpoint the migrated processes, a checkpointing daemon, called chpoxd, runs in the background. It reads from /etc/phantomos.cnf the value of the checkpointing interval. It then gets the list of processes registered for checkpointing, and checkpoints each process after the checkpointing interval. The checkpoints are saved in /tmp/phantomos. The names of the checkpoints are of the form pid.dump, e.g. a process with pid 642 would be checkpointing in the file /tmp/phantomos/642.dump. The process can be resumed from this checkpoint later. The following activity diagram shows how chpoxd works.



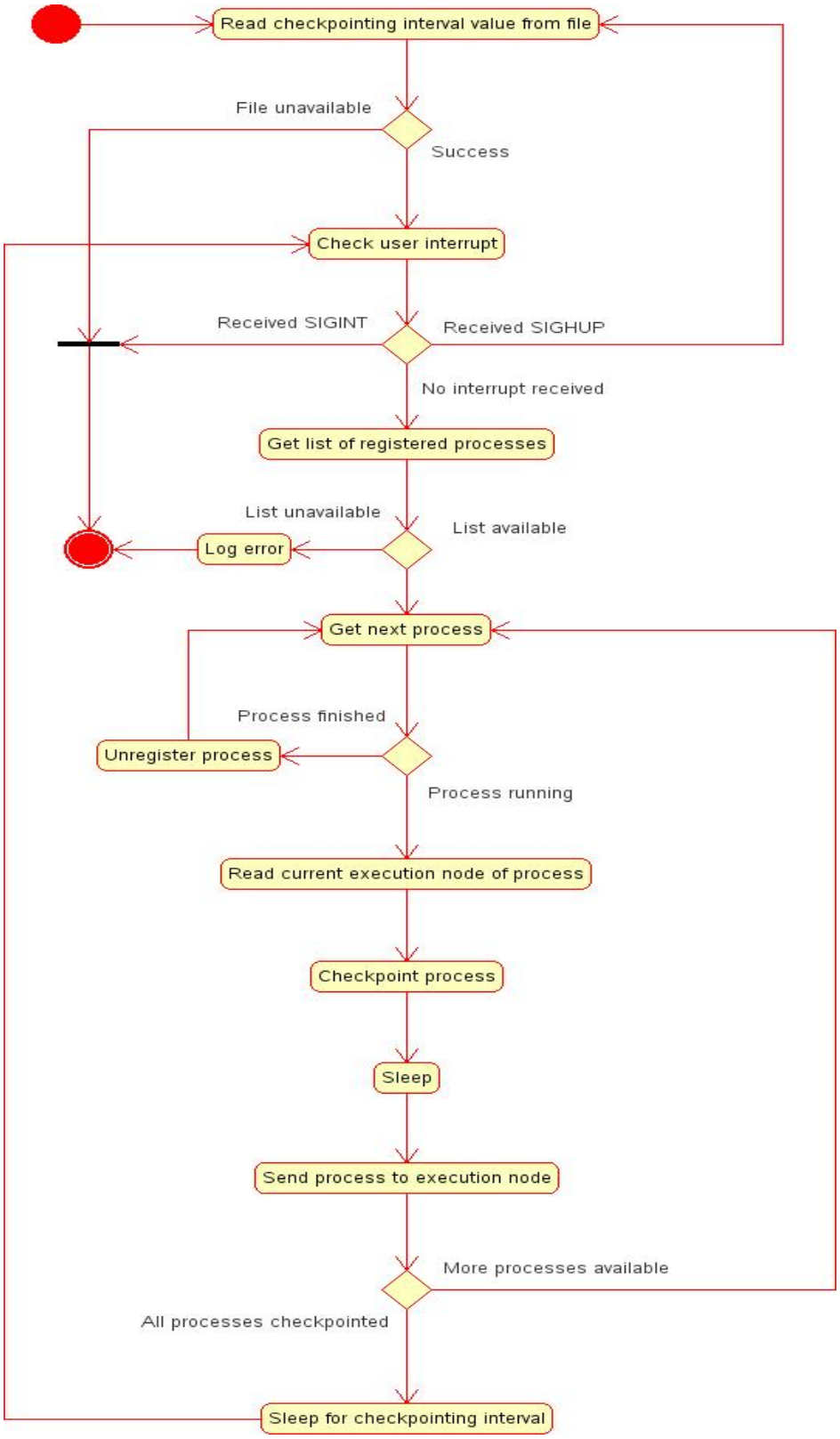



**Figure 5.1 Chpoxd Activity Diagram**

Chpoxd is written like any standard Linux daemon. When it runs, it writes it pid to /var/run/chpox.pid. The daemon is written so as to handle two kinds of signals. When it receives SIGINT, it does a clean exit, deleting /var/run/chpoxd.pid. When it receives SIGHUP, it rereads the value of the checkpointing interval from the file /etc/phantomos.cnf. It then uses the new value from that point onwards.

To checkpoint the process, the process has to be recalled to the home node. This is so because in order to checkpoint the process correctly, some kernel structures have to be written to disk along with the complete address space of the process. Therefore, the process is recalled to get a consistent snapshot of its address space and kernel structures. The daemon is responsible remembering what node the process was recalled from and then sending the process back to it.

The last job of chpoxd is to check if a particular process has completed. If so, it must unregister the process so as no future attempts to checkpoint it are made.

# Chapter 6

# 6. QoS Management

## 6.1 Introduction

Existing Grid Middleware delegate resource management to the cluster middleware. Modern Cluster middleware provide "all or nothing" resource management, either a node is completely available for processing, or its not. There is no way for individual resource users to control the amount of resources each machine should offer to the Grid. Fine grained resource control is not an issue in dedicated Grid systems, such as those involved in existing Grids. However in decentralized user or Enterprise Grids, participating nodes may be involved in multiple tasks, some of those might be time-critical hence a minimum level of QoS must be ensured in individual nodes. To allow for fine grained control over a user's resources, PhantomOS makes use of a QoS management module for local computations.

## 6.2 QoS Management Module

This module is used to force migration of processes which exceed the user defined QoS limits. The QoS limitations of the users are defined in an XML file. The QoS module is implemented as a kernel module which monitors real time resource usage of processes, with support for multi-core processors, and a daemon service to track trends of resource usage over time in order to make intelligent decisions about processes which are exceeding administrator defined resource limitations. So far QoS mechanism provides

support for controlling CPU and Memory utilization; additionally networking usage will be provided in future releases. The QoS can be turned off, if the PhantomOS node is to be used as part of a dedicated Grid.

The following figure shows how the QoS daemon works monitors the resource usage of each process.

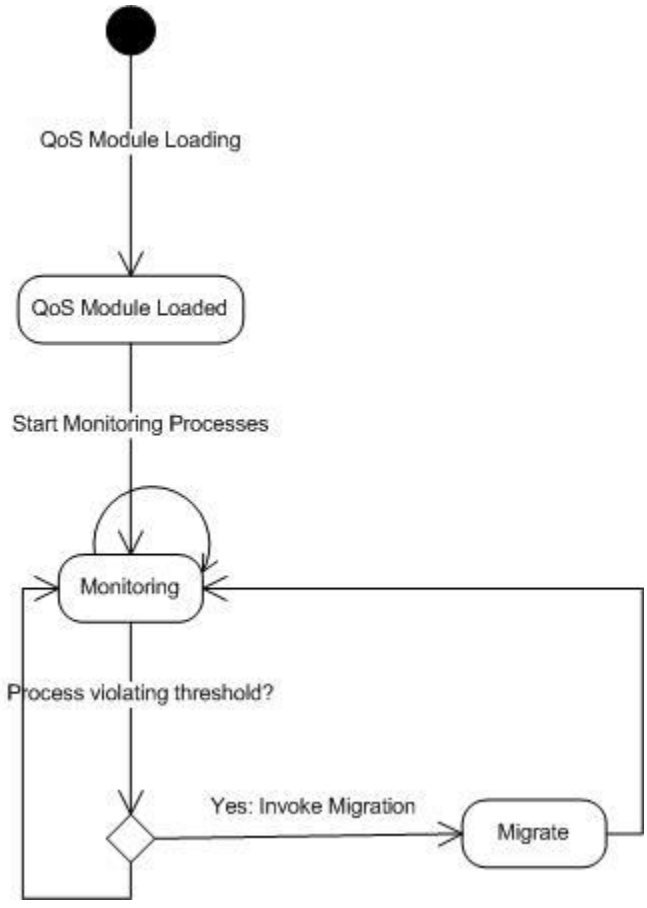



**Figure 6.1: QoS Management UML Activity Diagram**

# Chapter 7

# 7. Heart Beat Monitoring

## 7.1 Introduction

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 PhantomOS, in monitoring node churn rate and process level node activity. PhantomOS heartbeat monitoring is completely decentralized except for the HB monitor which monitors node churn rate.

## 7.2 Node Churn Heart Beat Monitoring

Node Churn HB Monitoring is required by the Super peer to maintain quality of service in the Node. It uses the HB_MON table to keep track of the nodes which have signed into a PhantomOS cluster. The work flow of the node churn HB Monitor is as follows:

The steps to the node churn heart beat monitor are described: The Super peer loads the heart beat monitor, which starts reading from the HB_MON table for IPs which are listed there. The heart beat monitor sequentially invokes the isAlive() method exposed by the web service interface of the resource broker.

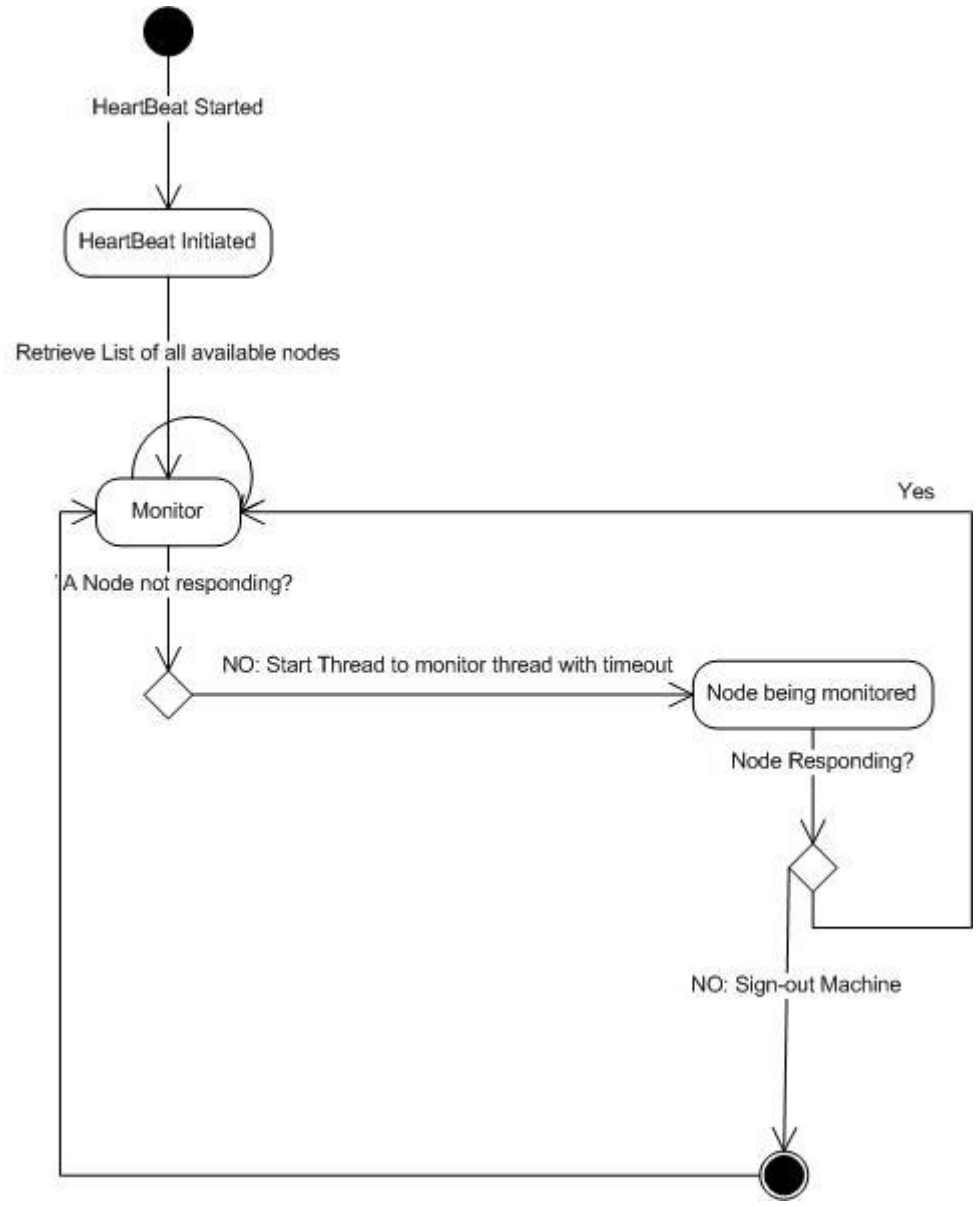



**Fig 7.1 Heart Beat Monitoring Activity Diagram**

Once the resource broker responds the HB monitor continues to the next node. If however the node fails to respond during a specific time, a thread is spawned which then closely monitors the node after a 60 sec timeout. If the node responds within the 60 sec timeout the node accepted as being part of the Grid. However if it fails to respond within

the said time, then the node is signed-off and will no longer be entertained for future resource requests.

The Node churn HB monitor was developed in python, as web service.

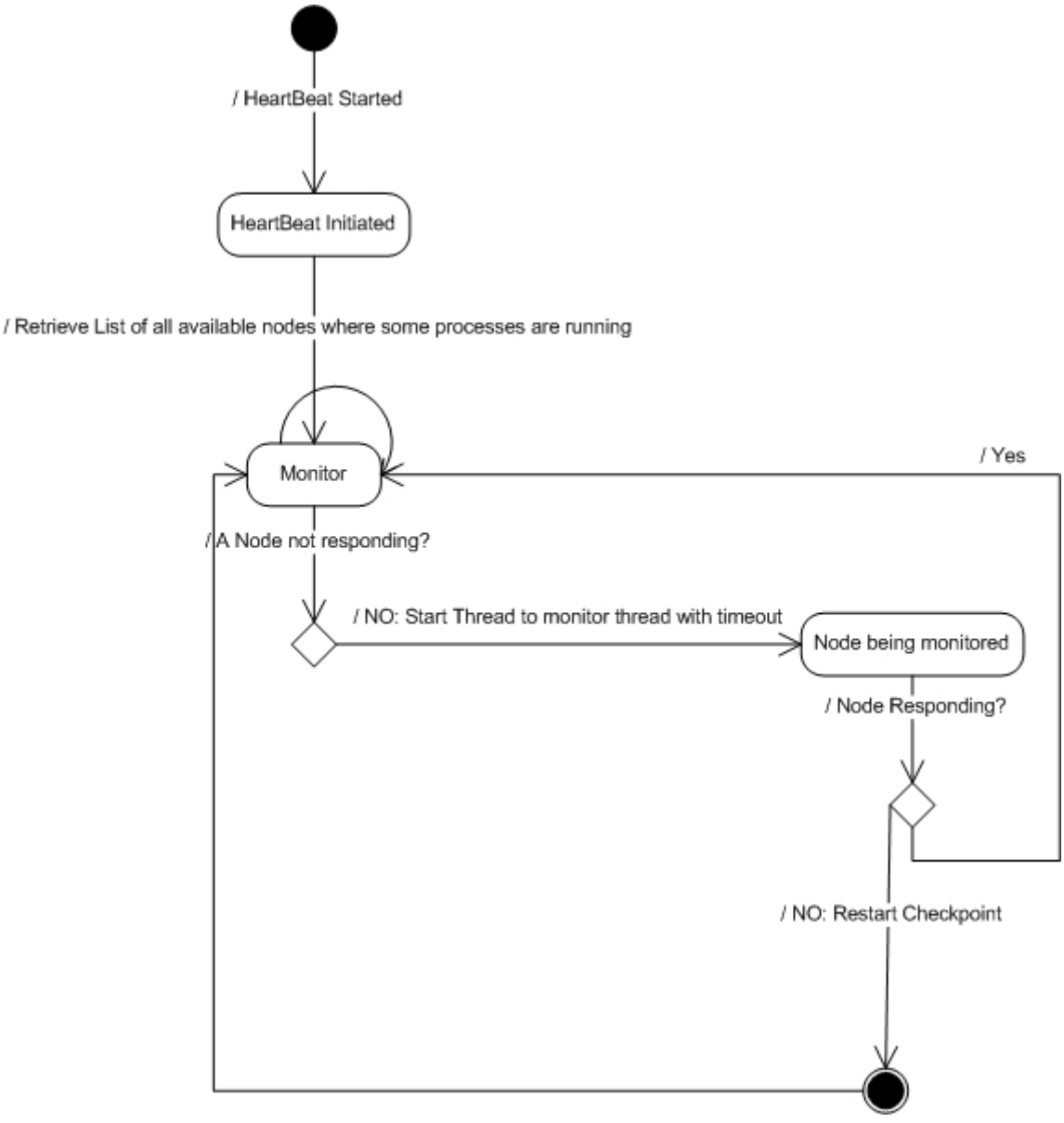



**Figure 7.2 Process Level Heart Beat Monitoring Avtivity Diagram**

Process Level HB monitoring is used to monitor eventual crashes of nodes while executing a processes. The concept is similar to that of the node churn level heart beat monitor how ever, the difference is in the way node crashes are handled instead of signing off the machine from the Grid, since it is not the super peer, once it detects that a machine which is running a process, and it is no longer available, the latest checkpoint will be retrieved and executed locally. The checkpoints are kept in /tmp/phantomos.

## 7.3 Summary

Heart Beat Monitoring is an important component in maintaining fault tolerance and stability in a PhantomOS 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. Discovery Service

## 8.1 Introduction

There are various approaches used for resource discovery which have been widely described in the literature [9]. In most of the existing Grid middleware resource discovery is handled in a centralized and/or hierarchical manner. For example, gLite 3.0 and Globus Toolkit (GT) 2.0 uses MDS-2[12] which is built around the centralized index service GIIS. GT 4.0 uses an improved form of MDS 4.0 with minor changes to the underlying architecture. UNICORE too is built around a client-server approach. Most Grid discovery services have cluster level granularity, and depend on the cluster middleware for low level discovery. However for more fine-grained resource consumption, as is required for multithreaded interactive applications, virtual organizations (VO) created based on machine-level granularity are needed. A desktop Grid will have no cluster, in the contemporary sense of the word, and will be based on a machine level granularity principle. Dealing with machine level granularity in client server architecture greatly reduces the QoS of the system. On the other hand implementing a machine-level granularity system for all applications radically reduces the QoS of the Grid and increases the complexity in scheduling and resource brokering. In the light of all this, we can see that existing Grid middleware does not provide fine-grained resource discovery solutions for resource discovery in widely distributed systems for interactive applications.

However, at the same time, machine-level granularity resource discovery algorithms used in cluster level operating systems such as OpenMOSIX and popular cluster middleware such as Condor do not have scalable discovery systems, which are primarily multicast based.

Scalability is an essential requirement for any widely distributed system. There are numerous challenges in designing a scalable resource discovery service for such a widely distributed system dealing down to the machine level. Adopting a pure peer to peer approach can radically reduce the QoS due to increased response times and larger search space for resource discovery. Another challenge for developing a resource discovery scheme for a desktop Grid is to tackle the volatile nature of machines, both in terms of availability and rapid processing load changes. Existing discovery services do not cater for such dynamic environments, as the resources they have to support are dedicated. Information dissemination or other schemes will no longer give the true resource status and will also produce high network traffic.

The authors believe that the Grid OS could serve as a means for the convergence of both Grid and P2P environments [16]. It could provide the necessary infrastructure for P2P environments to be introduced into a greater number of domains and move beyond simple data sharing. It would also enable Grid environments to adopt the scalability of P2P environments. The convergence of both technologies will thus lead to more ubiquitous deployment of distributed applications.

## 8.2. Proposed Scheme and Architecture

Our proposed discovery scheme for Grid OS is an enhancement over Mastroianni et al. [29]. 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.

We introduce a two-tier based super peer architecture: 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 3. 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.

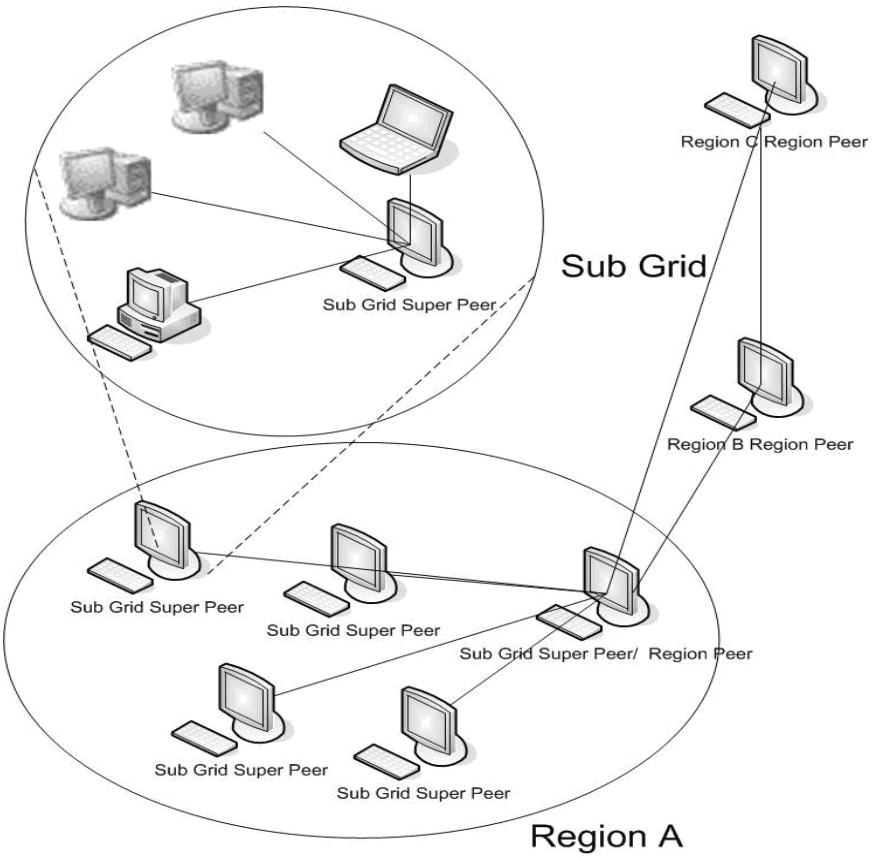



**Figure 8.1 Two tier super peer architecture**

The whole concept of a 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.

### 8.3 Resource discovery at the Intra-Sub Grid level

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.

## 8.4 Resource Discovery at Intra and Inter-Region Level

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.

## 8.5 Resource Discovery for Resource Intensive Applications

Resource intensive applications are those applications which require more processing than any single machines can afford. The previous two subsections assume that the resources are being requested for a task which can be handled by a single machine. However, there are numerous domains that require computations which are beyond the capabilities of single machines. The two-tier super peer model supports these applications as well, given that the application designers can accurately define the resources they would require. If the required resources can be defined, the Grid OS can create Grid nodes with the help of SSI, by virtualizing resources of entire subgrids or creating temporary virtual subgrids specifically for those applications. Details of this approach will be outlined in future publications.

There are some important open research issues, which will be investigated in future related to the Grid OS discovery service, for example, which measure gives a true reflection of the cumulative power of a subgrid? Contemporary cluster middleware does not provide such mechanisms, hence in a Grid these things are handled manually by the administrators; the Globus toolkit provides a resource specification language (RSL)[19] which is used by cluster administrators to describe their resources. In a decentralized system, as in Grid OS, there will be no subgrid administrator hence automated methods need to be devised in order to guide the region peers in scheduling decisions. Other issues, which will be determined via appropriate simulations, include for example, what is the ideal number of nodes in a sub-grid? Determining the ideal number of nodes within a sub-grid leads to a tradeoff between QoS and size of the sub-grid. The larger a sub-grid the larger is the search space for potential resources hence leading to greater lag in scheduling decisions. Similarly what is the optimum size of a region in the second tier? Additionally we also need to investigate which is the most appropriate method of handling network address translation (NAT) and firewalls. Contemporary Grids rely on static IPs as they are dedicated resources. However, in the case of Grid OS, resources are not dedicated and most probably be used within homes and offices, many of which have NAT and firewalls. If these are not handled suitably a Grid can develop where nodes behind NAT and firewalls have the capability of accessing everyone else, however no one will be able to access them, thus opening the system up to potential abuse.

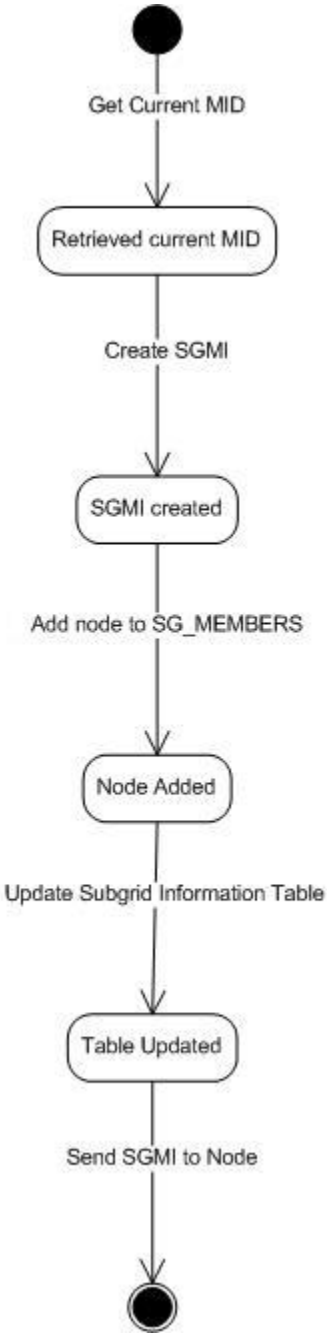



**Figure 8.2 Discovery Service Registration Process**

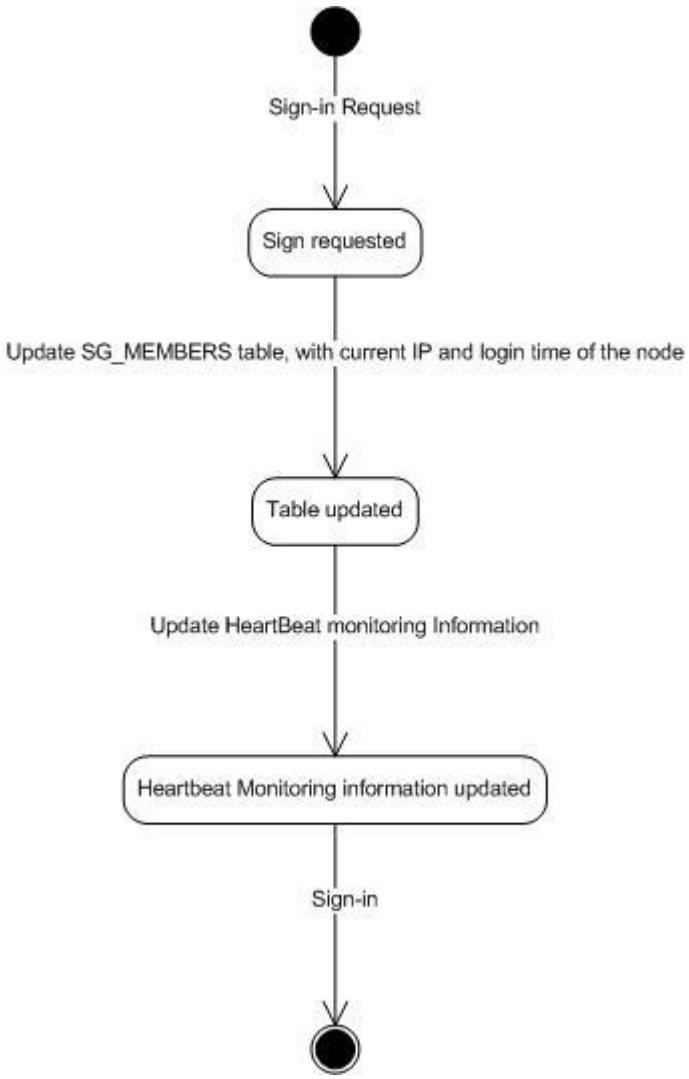



**Figure 8.3 Discovery Service Sign in Process**

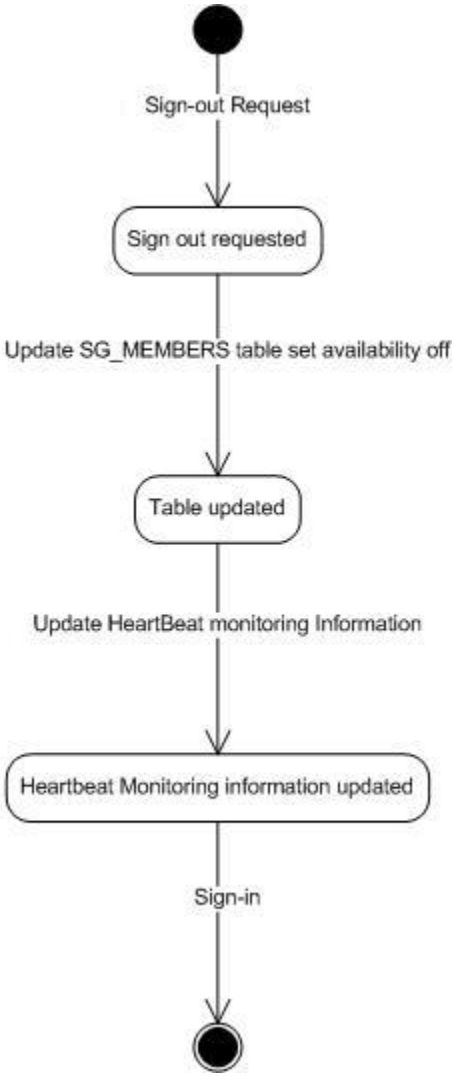



**Figure 8.4 Discovery Service Sign out Process**

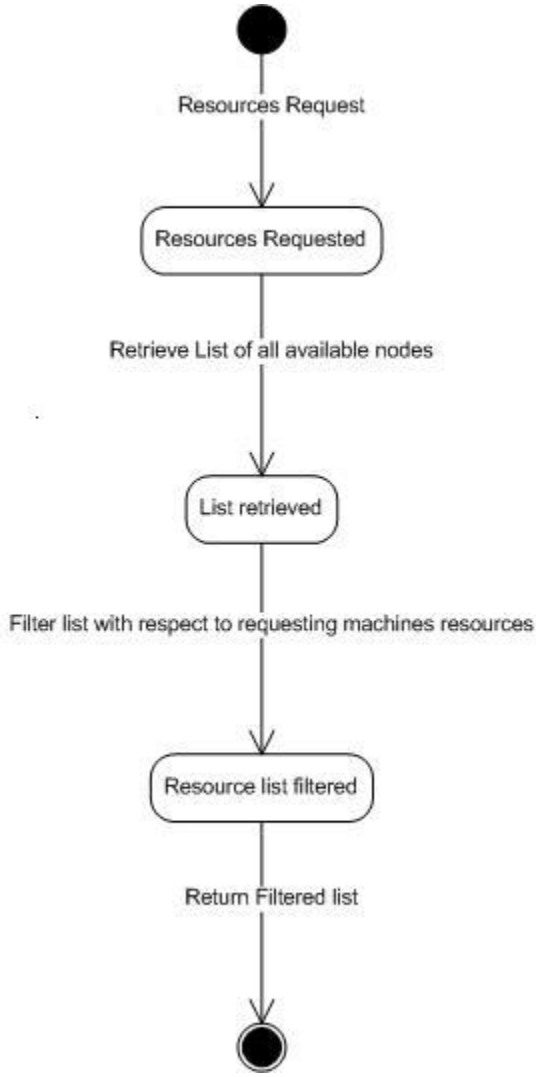



**Figure 8.5 Discovery Service Resource Request Process**

# Chapter 9

# 9. Analysis and Testing

## 9.1 Introduction

Numerous algorithms have been developed for resource brokering and scheduling 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. We will defer discussion of resource brokering for large-scale applications to future papers, and restrict our discussion here 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.

To devise an efficient resource brokering algorithm we have to consider the computing environment of the Grid OS:

- Machines will have highly dynamic availability, which renders unworkable the use of existing resource brokering algorithms which take dedicated resources into consideration;
- Machines will have rapid load changes, which means that resource broker which rely on static methods to broker resources will be insufficient;
- Machines will be separated by varying network links, hence scheduling decisions must consider the network as a resource, as it may impact on the runtime of the job;
- Machines will have heterogeneous hardware specification;
- In a desktop Grid there will likely by millions of nodes, hence centralized resource brokers are out of question, and distributed resource brokers have to be considered.

In the light of the above points, we can conclude that existing Grid resource brokering algorithms are insufficient for dynamic widely distributed environments such as in user-oriented Grids. Centralized resource brokers will have little use as they would have to be informed of availability changes in the nodes, which would cause a great deal of traffic. Distributed P2P resource brokers are more suitable in this environment [40]. Every node will have an independent resource broker, which will search and reserve resources for computations, and release the resources when computations complete. In order to limit the potential for all-to-all communication, the discovery service will be designed to compensate for that and for groups nodes in a subgrids based topology to manage scalability.

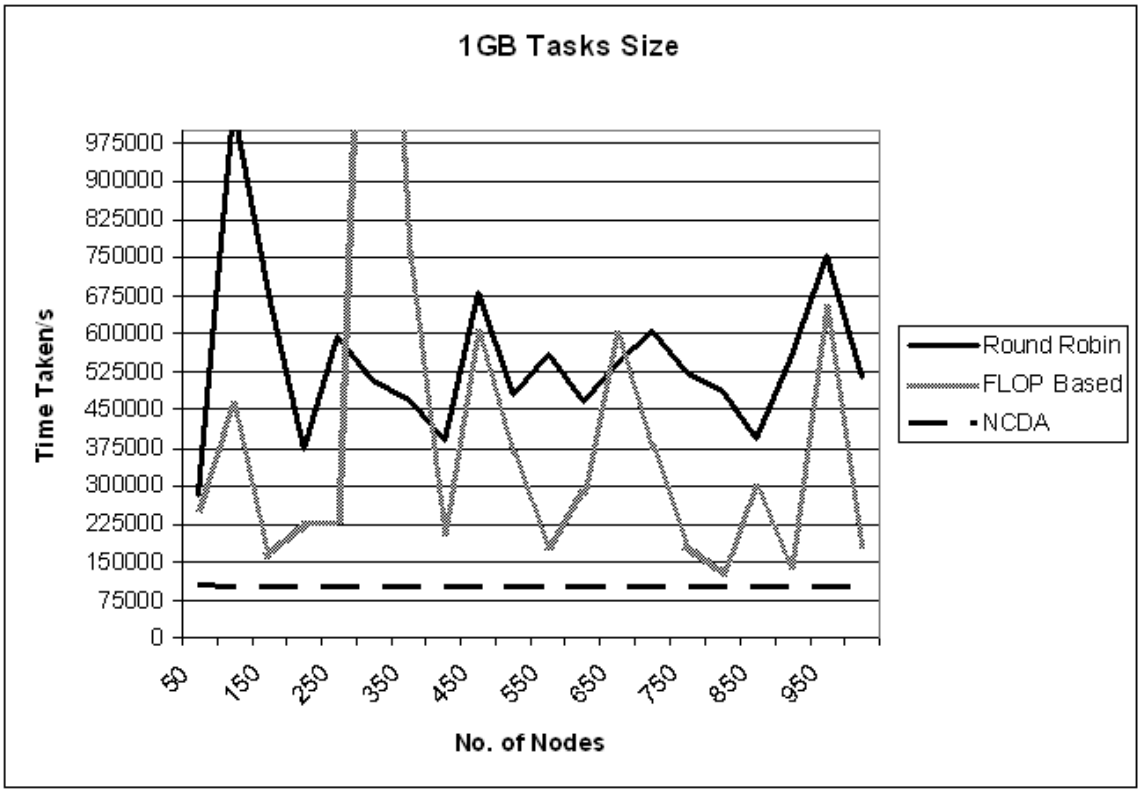
The resource brokering algorithm we have devised is a network compute and data aware algorithm (NCDA) and considers both network connectivity and computational capability in scheduling decisions. It is a variant of simple Condor ClassAds[34] based algorithm. ClassAds and related resource brokering and scheduling policies (from now abbreviated as FLOPS based algorithms) take the computational capability and the existing load into consideration and are popular in Cluster middleware. Nodes in clusters and clusters in contemporary Grids are usually linked via high-speed network links, often these high-speed network are only a small magnitude slower than the internal computer speeds, thus enabling the middleware to ignore the network in scheduling decisions. However when it comes to a system such as Grid OS, which tries to connect the resources of home and business users together, we cannot take the network for granted, and it has to be treated as a resource. Hence NCDA incorporates a measure of the network connectivity between nodes in resource brokering decisions. When a job is scheduled for migration to the Grid the resource broker requests members of the local sub grid for their current resource state. The member nodes respond to the request by dynamically creating resource descriptions and additionally at runtime try to determine the current real bandwidth from the machine which requested the descriptions. There many ways to determine the real bandwidth between two nodes, however there are limitations to each, therefore we base our bandwidth determination mechanism on the time-to-live (TTL) of of Internet control message protocol (ICMP) packets. Many organizations block ICMP traffic, hence in future we may support multiple methods of calculating bandwidth. Other methods such as using the simple network management protocol (SNMP) to determine bandwidth are

not feasible in most cases. Besides these two techniques there a plethora of bandwidth intrusive techniques, such as Iperf[22].

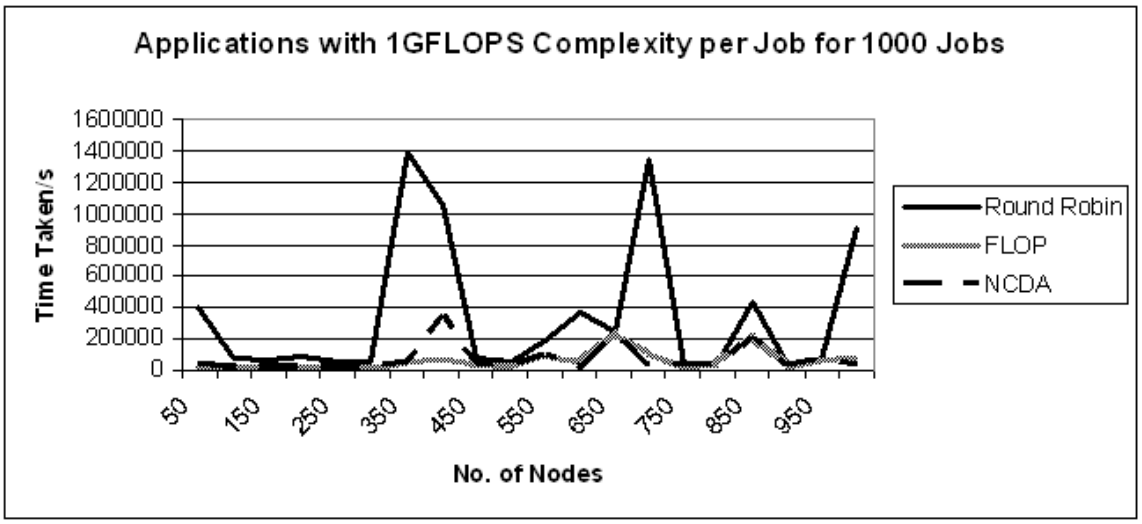
NCDA was benchmarked against the FLOPS based algorithms, and the round robin algorithm, which is popular in homogenous distributed computing environments and is implemented in some Grid middleware, most notably the Chimera VDS Sphinx Scheduler [20]. The three algorithms were subjected to different task types which range from compute intensive applications, network intensive applications and hybrid applications, which were both compute and network intensive. We define the performance of an algorithm as the quality of the resource selection; if an algorithm has "high performance" we mean that it has the capability to select best possible resources for a job. The round robin algorithm which has been tested is not pre-emptive, as is the case in operating systems. In the Grid round robin jobs are scheduled successively to different machines in a specific order without regards to the resource load. Round Robin algorithm is ideal for environments which have homogeneous hardware resources, or clusters which have equivalent computer power.

We have used SimGrid 3.1[26] to implement the algorithms, and test them in randomly generated computing platforms of 50 to 1000 nodes with random network links ranging from 56kps (typical dial-up connection) to 10MBps, the computational capability of the nodes generated ranges from 10KFLOPS to 100MFLOPS. Applications were generated for each of the three categories: compute intensive, network intensive and hybrid applications, in each instance 1000 jobs were scheduled. Compute intensive applications

required a processing of 1giga floating point operations each, whereas network intensive applications caused network traffic of 1 GB each. Hybrid applications required both processing of 1 GFLOPS and caused 1 GB network traffic each. The graphs in figures 19, 20 and 21 show the performance of each algorithm in each instance of scheduling.

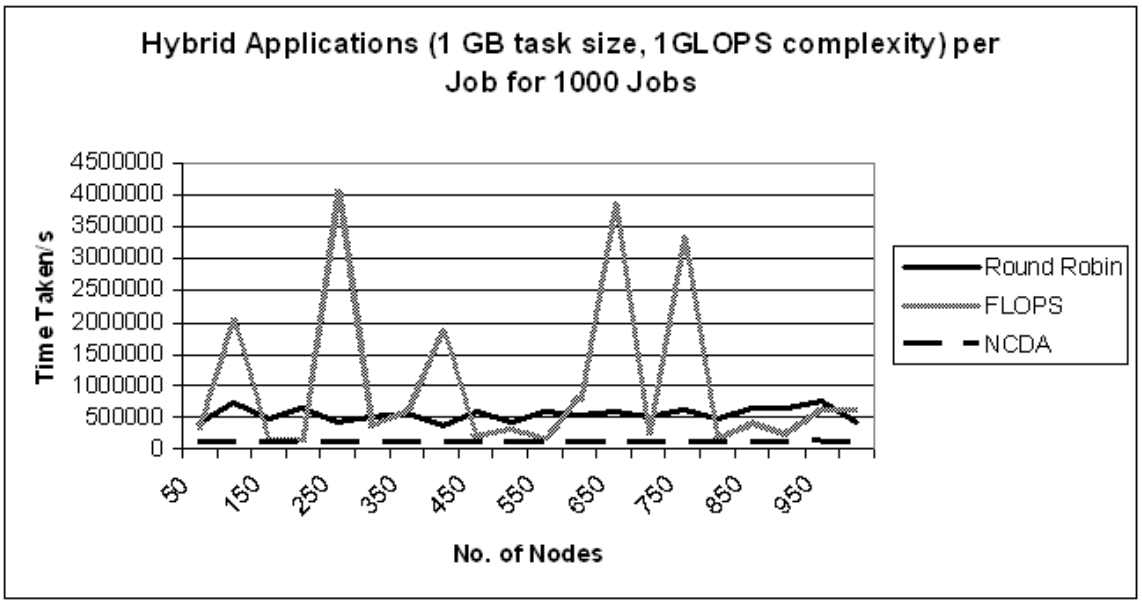



**Graph 9.1 Performance of algorithms for network intensive applications**




**Graph 9.2 Performance of algorithms for compute intensive applications**

Analyzing the results we can see that NCDA provides good performance in environments where network connectivity matters, such as for network intensive (Figure 19) and hybrid applications (Figure 21). It however underperforms for computational complex tasks (Figure 20) in which case the FLOPS based algorithms have the best performance. In the results shown above, the overhead in scheduling is not shown. NCDA has a O(n) scheduling complexity, which means that as the number of the nodes increases the time to perform a scheduling decision increases proportionally. Additionally there are some overheads involved in determining the bandwidth between the nodes. To handle this we have designed our discovery topology around a two-tier super peer based architecture where subgrids are formed on the basis of small round trip time (RTT) to limit the effects of network latencies and scale




**Graph 9.3: Performance of algorithms for both network and compute intensive applications**

## 9.2 Simulation Code

```
{
    slaves_count = argc - 4;
    slaves = calloc(slaves_count, sizeof(m_host_t));

    for (i = 4; i < argc; i++) {
      slaves[i-4] = MSG_get_host_by_name(argv[i]);
      if(slaves[i-4]==NULL) {
        INFO1("Unknown host %s. Stopping Now! ", argv[i]);
        abort();
      }
    }
  }

  INFO1("Got %d slave(s) :", slaves_count);
  for (i = 0; i < slaves_count; i++)
   { INFO1("\t %s", slaves[i]->name);
INFO1("\t %f", MSG_get_host_speed(slaves[i]));
}
```

**SimGrid Algorithm for Round Robin**

```
{                   slaves_count = argc - 4;
    slaves = calloc(slaves_count, sizeof(m_host_t));

    for (i = 4; i < argc; i++) {
      slaves[i-4] = MSG_get_host_by_name(argv[i]);
      if(slaves[i-4]==NULL) {
        INFO1("Unknown host %s. Stopping Now! ", argv[i]);
        abort();
      }
    }
  }

/*comparison*/
if( MSG_get_host_speed(slaves[2]) > MSG_get_host_speed(slaves[3]))
{
INFO1("This host is BIGGER %s",MSG_host_get_name(slaves[0]));
}else
{
INFO1("This host is BIGGER %s",MSG_host_get_name(slaves[1]));
}

  INFO1("Got %d slave(s) :", slaves_count);
  for (i = 0; i < slaves_count; i++)
   { INFO1("Slave: \t %s", slaves[i]->name);
INFO1("\t %l", MSG_get_host_speed(slaves[i]));
}
  INFO1("Got %d task to process :", number_of_tasks);

  for (i = 0; i < number_of_tasks; i++)
    INFO1("\t\"%s\"", todo[i]->name);

  for (i = 0; i < number_of_tasks; i++) {

int bestnode = getBest();

  INFO2("Sending \"%s\" to \"%s\"",
                  todo[i]->name,
                  slaves[bestnode]->name);
    MSG_task_put(todo[i], slaves[bestnode],PORT_22);
      load[bestnode] = MSG_task_get_compute_duration(todo[i]);
    INFO0("Send completed");
  }

  INFO0("All tasks have been dispatched. Let's tell everybody the
computation is over.");
  for (i = 0; i < slaves_count; i++)
    MSG_task_put(MSG_task_create("finalize", 0, 0, FINALIZE),
                 slaves[i], PORT_22);

  INFO0("Goodbye now!");
  free(slaves);
  free(todo);
  return 0;
} /* end_of_master */
```

## SimGrid Algorithm for FLOP Rating

```
int getBest()
{
int best = 0;
int i=0;
for( i=1;i<size;i++)
{
if( MSG_get_host_speed(slaves[best])-load[best] <
MSG_get_host_speed(slaves[i])-load[i])
{
best = i;
}
}
INFO1("Returning node: %d",best);
return best;
}
```

**Calculation of Best Node in FLOP Rating**

```
{
    slaves_count = argc - 4;
    slaves = calloc (slaves_count, sizeof (m_host_t));
    bw = calloc (slaves_count, sizeof (double));
 load = calloc (slaves_count, sizeof (double));

   //load = bw;

//tasks dispatched structure initialization code


    int ui = 0;
    for (ui = 0; ui < slaves_count; ui++)
      {
      load[i] = 0;
      }

    for (i = 4; i < argc; i++)
      {
      slaves[i - 4] = MSG_get_host_by_name (argv[i]);
      if (slaves[i - 4] == NULL)
        {
          INFO1 ("Unknown host %s. Stopping Now! ", argv[i]);
          abort ();
        }
      }
  }

  INFO1 ("Got %d slave(s) :", slaves_count);
  for (i = 0; i < slaves_count; i++)
    {
      INFO1 ("Slave: \t %s", slaves[i]->name);
      INFO1 ("\t %l", MSG_get_host_speed (slaves[i]));
    }
  INFO1 ("Got %d task to process :", number_of_tasks);

  for (i = 0; i < number_of_tasks; i++)
    INFO1 ("\t\"%s\"", todo[i]->name);

  struct taskd tasksd[5000];
  structsptr = tasksd;

  double str;
  int f;
```

```
int id=0;
slavecount=slaves_count;
for(id=0;id<slavecount;id++)
{
structsptr[id].count=0;
}

for(id=0;id<slavecount;id++)

{
INFO1("SD %d",structsptr[id].count);
}

  for (i = 0; i < number_of_tasks; i++)
    {
      int bestnode =
      getBest (MSG_task_get_compute_duration (todo[i]),
             MSG_task_get_data_size (todo[i]));

      INFO2 ("Sending \"%s\" to \"%s\"",
           todo[i]->name, slaves[bestnode]->name);
      MSG_task_put (todo[i], slaves[bestnode], PORT_22);
      tasksd[bestnode].tasks[tasksd[bestnode].count++] = i;
      INFO0 ("Send completed");
    }

  INFO0
    ("All tasks have been dispatched. Let's tell everybody the
computation is over.");
  for (i = 0; i < slaves_count; i++)
    MSG_task_put (MSG_task_create ("finalize", 0, 0, FINALIZE),
              slaves[i], PORT_22);

  INFO0 ("Goodbye now!");
  free (slaves);
  free (todo);
  return 0;
}                       /* end_of_master */
```

**SimGrid Algorithm for NCDA Rating**

## 9.3 Summary

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

# Chapter 10

# 10. Future Work

## 10.1 Future Work and Conclusion

Grid computing, despite having made huge progress during the last decade, is not pervasive as has been promised over recent years. This lack of adoption can be traced to some barriers which have resulted from the strong focus of Grid computing research on the middleware approach and on a particular community of largely scientific and engineering users. This leads to the introduction of some limitations in the existing Grid infrastructure in terms of lack of interactive application support, resource discovery in highly dynamic environments and scalable topologies for Grid with loosely coupled dynamic clusters. In addition, the infrastructure required to manage Grids and the steep learning curve associated with maintaining contemporary Grid middleware would discourage potential user adoption of Grid computing. PhantomOS aims at the development of a pervasive general purpose Grid computing platform for both common and existing Grid users by converging user-centric computing with eScience-centric Grid computing. The main contribution of this project is to define the components of the Grid OS which aim at removing most of the technical barriers to Grid computing for common and business users while highlighting open research issues, which will be tackled during the ongoing course of the Grid OS project. This thesis presents a synopsis of the implementation which has been carried out as part of our degree project and has

established mechanisms to transparently grid-enable interactive desktop applications, fine grained distributed resource brokering, a two-tier super peer model based discovery topology aimed at converging centralized Grid computing with the decentralized peer to peer architecture, leading to a potentially high QoS, scalable, self-organizing and fault tolerant Grid.

Future work is forseen in two directions. Initially we are targeting the creation of technologies in the PhantomOS to support user-centric Grid computing. Future work in this area includes the development of a lightweight virtualization engine based on KVM to provide a security framework to the OS, extending fault tolerance to encompass the capability to enable self-reorganization of the topology in response to some failure at any level and shared memory for enabling thread migrations. The other research direction focuses on making the PhantomOS relevant to existing Grid users, and relieving them of some of the drawbacks of the middleware highlighted in this paper. Future work in this area includes embedding the capability for interoperability with the existing and emerging Grid infrastructure by making the system compliant to emerging standards in Grid computing, as approved by the Open Grid Forum, and supporting the virtualization of resources for resource intensive applications with the help of SSI to support existing Grid applications and to enable more fine-grained resource management in Grids.

# BIBLIOGRAPHY

**[1]** A. Ali et al., From Grid Middleware to a Grid Operating System, International Conference on Grid and Cooperative Computing, Oct 21-24, 2006, Changhsa, China.

**[2]** D. P. Anderson, BOINC: A System for Public-Resource Computing and Storage, 5th IEEE/ACM International Workshop on Grid Computing, November 8, 2004, Pittsburgh, USA.

**[3]** Apple xGrid, http://www.apple.com/acg/xgrid/

**[4]** A. Barak and O. La'adan, The MOSIX Multicomputer Operating System for High Performance Cluster Computing, Journal of Future Generation Computer Systems, Vol. 13, No. 4-5, pages 361-372

**[5]** I. Blanquer et al., Clinical Decision Support Systems (CDSS) in GRID Environments, 3rd International HealthGrid Conference, Oxford April 2005, IOS Press Studies in Health Technology and Informatics.

**[6]** B. Bode et al., The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters, Usenix Conference, Atlanta, GA, October 12-14, 2000.

**[7]** B. Calder, A. Chien, J. Wang & D. Yang, The Entropia Virtual Machine for Desktop Grids, International Conference on Virtual Execution Environments, June 2005.

**[8]** Case Studies & Success Stories, Enterprise Grid Alliance, http://www.gridalliance.org/en/resources/success_stories.asp

**[9]** S. Chaisiri & P. Uthayopas, Survey of Resource Discovery in Grid Environments, Proceedings of the IEEE Workshop on Experimental Distributed Systems, 1990.

**[10]** J. Chu et al., A Distributed Paging RAM Grid System for Wide-Area Memory Sharing, Proceedings of 20th International Parallel and Distributed Processing Symposium, Rhodes Island, Greece, 2006,25-29 April 2006.

**[11]** Condor Limitations
http://www.cs.wisc.edu/condor/manual/v6.2/1_4Current_Limitations.html

**[12]** K. Czajkowski, S. Fitzgerald, I. Foster & C. Kesselman, Grid Information Services for Distributed Resource Sharing, Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

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

**[14]** Essential Facts 2006: About the Computer and Video Game Industry, Entertainment Software Association

**[15]** I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, 11(2):115-128, 1997.

**[16]** I. Foster & A. Iamnitchi, On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), February 2003, Berkeley, CA.

**[17]** J. Freund, et al., Health-e-Child: An Integrated Biomedical Platform for Grid-Based Pediatrics, Studies in Health Technology & Informatics # 120, pp 259-270 IOS Press, 2006.

**[18]** Gartner Group, Gartner Predicts: Future of IT, Symposium/ITxpo, Cannes, November 2003.

**[19]** Z. Huang, L. Gu, B Du,& C. He, Grid Resource Specification Language based on XML and its Usage in Resource Registry Meta-service, Proceedings 2004 IEEE International Conference on Services Computing, 2004 (SSC 2004), 15-18 Sept. 2004 pages 467 – 470.

**[20]** J. In et al., SPHINX: A Fault-Tolerant System for Scheduling in Dynamic Grid Environments, Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, April 04-08 2005, page 12b.

**[21]** The Information Societies Technology Project: MammoGrid – A European Federated Mammogram
Database Implemented on a Grid Infrastructure, EU Contract IST 2001-37614 http://mammogrid.vitamib.com.

**[22]** Iperf Network Measurement Tool, http://dast.nlanr.net/Projects/Iperf/

**[23]** H. Jiang & V. Chaudhary, Compile/Run-time Support for Thread Migration, Proceedings of 16th International Parallel and Distributed Processing Symposium, Fort Lauderdale, Florida, April 15-19, 2002.

**[24]** K. Krauter, R. Buyya, & M. Maheswaran, A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing, Software: Practice and Experience (SPE), ISSN: 0038-0644, Volume 32, Issue 2, Pages: 135-164, Wiley Press, USA, February 2002.

**[25]** E. Laure et al., Middleware for the Next Generation Grid Infrastructure, Proceedings of the Computing in High Energy Physics Conference, pages 826, 2004.

**[26]** A. Legrand, L. Marchal & H. Casanova., Scheduling Distributed Applications: The SimGrid Simulation Framework, 3rd IEEE International Symposium on Cluster Computing and the Grid 2003, pages 138 – 145.

**[27]** M. Litzkow, M. Livny, & M. Mutka, Condor - A Hunter of Idle Workstations, Proceedings of the 8th Int. Conference of Distributed Computing Systems, June 1988, pages 104-111.

**[28]** M. Litzkow et al., Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System, University of Wisconsin-Madison Computer Sciences Technical Report #1346, April 1997

**[29]** 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 .

**[30]** D. S. Milojičić et al., Process Migration, ACM Computing Surveys, September 2000, pages 241 – 299.

**[31]** OpenMOSIX, http://openmosix.sourceforge.net

**[32]** R. Pennington, Terascale Clusters and the TeraGrid, 7th International Conference on High Performance Computing and Grid in Asia Pacific Region, Dec 16-19, 2002, pp. 407-413.

**[33]** M. Pourzandi, D. Gordon, W. Yurcik & G. A. Koenig, Clusters and Security: Distributed Security for Distributed Systems,1st International Workshop on Cluster Security 2005, Cardiff, UK.

**[34]** 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.

**[35]** D. Reed, I. Pratt et al., Xenoservers: Accountable Execution of Untrusted Code, IEEE Hot Topics in Operating Systems (HotOS) VII, March 1999.

**[36]** J. P. Ryan & B. A.Coghlan, B.A. SMG: Shared Memory for Grids, 3rd International Conference on Parallel and Distributed Computing Systems (PDCS'04), Boston, Nov. 2004.

**[37]** H. Takemiya, K. Shudo, Y. Tanaka & S. Sekiguchi, Constructing Grid Applications Using Standard Grid Middleware, The Eighth Global Grid Forum, June 25, 2003 .

**[38]** 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

**[39]** 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.

**[40]** 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

**[41]** 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