A
Dissertation
on

dgridOS – An approach to Grid enabled Operating System

Submitted in partial fulfillment of the requirement for the award of Degree of

MASTER OF ENGINEERING
(Computer Technology and Application)

Submitted by
DHIRAJ KUMAR SINGH
(University Roll No. 2001)

Under the Guidance of
Prof. D. ROY CHOUDHURY

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
BAWANA ROAD, NEW DELHI
(DELHI UNIVERSITY)

JUNE-2007

CERTIFICATE

This is to certify that the work that is being presented in this dissertation entitled "**dgridOS - An approach to Grid enabled Operating System**" submitted by **Dhiraj Kumar Singh** in the partial fulfillment of the requirement for the award of the degree of **Master of Engineering** in Computer Technology & Application, Delhi College of Engineering is an account of his work carried out under my guidance in the academic year 2006-2007.

This work embodies in this dissertation has not been submitted for the award of any other degree to the best of my knowledge.

Prof. D Roy Choudhury
Head of Department
Department of Computer Engineering
Delhi College of Engineering
Delhi

Dr. S C Gupta
Sr. Technical Director
National Informatics Center
Delhi

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extend my heartiest felt gratitude to everybody who helped me throughout the course of this project.

It is distinct pleasure to express my deep sense of gratitude and indebtedness to my learned supervisors Dr. S C Gupta and Prof. D Roy Choudhury for their invaluable guidance, encouragement and patient reviews. Their continuous inspiration only has made me complete this dissertation. Both of them kept on boosting me time and again for putting an extra ounce of effort to realize this work.

I am also thankful to my lab partners Kalpesh Kumar Meena, Minakshi Anand and Rajesh Kumar, for their consistent support and cheerups.

I would also like to take this opportunity to present my sincere regards to my teachers Prof. Goldie Gabrani, Mrs. Rajni Jindal, Dr. S. K. Saxena, Mr. Manoj Sethi and Mr. Rajeev Kumar for their support and encouragement.

I am grateful to my parents, brothers and sisters for their moral support all the time, they have been always around on the phone to cheer me up in the odd times of this work.

I am also thankful to my classmates for their unconditional support and motivation during this work. Living at DCE with them has been a lifetime experience for me, all the time we spend together enjoying life to its fullest, the birthday parties, placement parties, photo sessions and night outs discussing new topic or technology would remain with me forever.

I want to thank the members of Linux Forums, Grid Computing communities, Sourceforge.net. Last but not least, special thanks to the crowd who are active in the field of distributed computing on World Wide Web.

(Dhiraj Kumar Singh)
M.E. (Computer Technology and Application)
Department of Computer Engineering
Delhi College of Engineering, Delhi-42

ABSTRACT

This work is about the realization of a modern operating system with grid services as an integral part of it. In recent time the dominating species of distributed computing is Grid Computing, and till now this grid computing has been enabled through the use of middleware toolkits.

As the most frequent machine interacting system calls are bundled together to make the kernel of operating system, so as the more frequently used functions are bundled together in set of modules which can easily be integrated to the operating system kernel even at runtime. In this process the grid services which enables the sharing, selection, and aggregation of wide variety of geographically distributed resources can be integrated together inside the kernel of operating system as a module.

In the last few years, a number of exciting projects like Globus, Legion, and UNICORE developed the software infrastructure needed for grid computing. However, operating system support for grid computing is minimal or non-existent. Tool writers are forced to re-invent the wheel by implementing from scratch. This is error prone and often results in sub-optimal solutions. To address this and some more problems, an approach has been described to demonstrate a module based approach to grid enabled operating systems.

In order to develop an integrated, GRID-enabled OS the free availability of the source code is mandatory. This eventually makes Linux (and other Open Source Operating Systems) the ideal platform for GRID computing. Linux already has a strong position in GRID research, owing to the fact that GRID computing inherits many of its features and ideas from clustering.

The core functional infrastructure is analyzed through different grid products in use which helped in isolation of operating system configuration that facilitate grid computing and common grid services provided by grid software. Based on the identified functionalities, modules and sub-modules description and interconnection is explored. Finally a setup execution plan is defined which visualizes the dgridOS structure and operation.

Just like it was the case in the beginning of the World Wide Web, there are currently many special purpose GRIDs, which usually use the Internet for data transportation. It'll take its time until these GRIDs grow together and form a World Wide GRID, but the ultimate goal is a global, standardised infrastructure for transparent execution of compute jobs across network boundaries and that is the evolution of Grid enabled operating system.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 BACKGROUND

As modern computing continues to see technological improvements in raw computing power, storage capability and communication, the demands placed on these resources continues to grow. Despite these improvements, there exist many situations where computational resources fail to keep up with the demands placed on them. This trend created the desire to be able to share computational resources both within an organization and with external organizations. Resource sharing is beneficial since it allows one to take advantage of the power of multiple resources in order to achieve a single goal and provides a method for harnessing the power of underutilized resources. Issues surrounding resource sharing form the basis for grid computing, an evolution of wide-area parallel and distributed computing at a multi-institutional scale. A grid is an infrastructure that allows the sharing, selection and aggregation of heterogeneous, geographically dispersed resources that can be owned and operated by different organizations. Distributed Computing is not a new paradigm. But up until a few years ago networks were too slow to allow efficient use of remote resources. But as the bandwidth of high-speed WAN's today even exceeds the bandwidth found in the internal links of commodity computers, it becomes inevitable that distributed computing is taken to a new level. It now becomes feasible to actually think of a set of computers coupled through a high-speed network as one large computational device.

1.2 MOTIVATION

My earlier project work was establishment of a working cluster[6] in our computer department. In the process of clustering of resources and their optimal use, it was discovered that cluster of clusters can make a grid and as cluster services are integrated inside the operating system then with few modifications a grid enabled operating system can be presented for general purpose use. With few encouragements from my guide and friends, it was decided to design a grid enabled operating system.

1.3 INTRODUCTION TO GRID COMPUTING[7][10][17]

1.3.1 GRID DEFINITION

The grid computing model of computing emerged recently as a new field distinguished from traditional distributed computing because of its focus on large-scale resource sharing and innovative high-performance applications.

The definition of Grid computing in Ian Foster's[11] three point check list that defines a Grid as a system that:

- coordinates resources that are not subject to centralized control,
- uses standard, open, general-purpose protocols and interfaces,
- delivers nontrivial Qualities of Service.

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing. Grid computing is a new IT architecture that produces more resilient and lower cost enterprise information systems. With grid computing, groups of independent, modular hardware and software components can be connected and rejoined on demand to meet the changing needs of businesses.

1.3.2 GRID BENEFITS[22]

Exploiting underutilized resources

The machine on which the application normally run might be busy due to an unusual peak in activity, then the job can be run on an idle machine elsewhere in grid. There are two prerequisites for this scenario. First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application (i.e. applications are grid enabled). A batch job that spends a significant amount of time processing a set of input data to produce an output set is perhaps the most ideal and simple use for a grid.

Parallel CPU capacity

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. A CPU intensive grid application can be thought of as many smaller "sub jobs," each executing on a different machine in the grid. To the extent that these sub jobs do not need to communicate with each other, the more "scalable" the application becomes. Perfectly scalable applications will, for example, finish10 times faster if it uses 10 times the number of processors.

Applications

Automatic transformation of applications is a science in its infancy. There are some practical tools that skilled application designers can use to write a parallel grid application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability. But still these software can target some applications to support some degree of scalability.

Virtual resources and virtual organizations for collaboration

Grid computing offers important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources. Sharing is not limited to files, but also includes many other resources, such as equipment, software, services, licenses, and others. These resources are "virtualized" to give them a more uniform interoperability among heterogeneous grid participants. The participants and users of the grid can be members of several real and virtual organizations. The grid can help in enforcing security rules among them and implement policies, which can resolve priorities for both resources and users.

Access to additional resources

Some machines may have expensive licensed software installed that the user requires. His jobs can be sent to such machines more fully exploiting the software licenses. Some machines on the grid may have special devices. Most of us have used remote printers, perhaps with advanced color capabilities or faster speeds. In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services.

Resource balancing

A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. An unexpected peak can be routed to relatively idle machines in the grid. If the grid is already fully utilized, the lowest priority work being performed on the grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work. When jobs communicate with each other, the Internet, or with storage resources, an advanced scheduler could schedule them to minimize communications traffic or minimize the distance of the communications. A grid provides excellent infrastructure for brokering resources.

Reliability

Grid management software can automatically resubmit jobs to other machines on the grid when a failure is detected. If there is a power or other kind of failure at one location, the other parts of the grid are not likely to be affected. In critical, real-time situations, multiple copies of the important jobs can be run on different machines throughout the grid, their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering. In principle, most of the reliability attributes achieved using hardware in today's high availability systems can be achieved using software in a grid setting in the future. It also provides mechanism for authorization, authentication for users making it more secure.

Management

It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization. With the larger view a grid can offer, it becomes easier to control and manage when hardware might be underutilized while another project finds itself in trouble. Aggregating utilization data over a larger set of projects can enhance an organization's ability to project future upgrade needs. When maintenance is required, grid work can be rerouted to other machines without crippling the projects involved.

1.3.3 TYPES OF RESOURCES IN GRID

Computation

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

Storage

A grid providing an integrated view of data storage is sometimes called a "data grid." Storage can be memory attached to the processor or it can be "secondary storage" using hard disk drives or other permanent storage media. Memory attached to a processor usually has very fast access but is volatile. It would best be used to cache data or to serve as temporary storage for running applications. Secondary storage in a grid can be used in interesting ways to increase capacity, performance, sharing, and reliability of data.

Communication

Another important resource of a grid is data communication capacity. This includes communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of data to be processed and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid. External communication access to the Internet, for example, can be valuable when building search engines. Machines on the grid may have connections to the external Internet in addition to the connectivity among the grid machines. When these connections do not share the same communication path, then they add to the total available bandwidth for accessing the Internet. Redundant

communication paths are sometimes needed to better handle potential network failures and excessive data traffic.

Software and Licenses

The grid may have software installed that may be too expensive to install on every grid machine. Using a grid, the jobs requiring this software are sent to the particular machines on which this software happens to be installed. License management software keeps track of how many concurrent copies of the software are being used and prevents more than that number from executing at any given time. The grid job schedulers can be configured to take software licenses into account, optionally balancing them against other priorities or policies.

Special equipment, capacities, architectures, policies

Some software may be available on several architectures, for example PowerPC and x86, such software is often designed to run only on a particular type of hardware and operating system. Such attributes must be considered when assigning jobs to resources in the grid. In some cases, the administrator of a grid may create a new artificial resource type that is used by schedulers to assign work according to policy rules or other constraints. For example, some machines may be designated to only be used for medical research.

1.3.4 APPLICATIONS AND JOBS

Usually we use the term application as the highest level of a piece of work on the grid. However, sometimes the term job is used equivalently. Applications may be broken down into any number of individual job, those, in turn, can be further broken down into sub jobs. A grid application that is organized as a collection of jobs is usually designed to have these jobs execute in parallel on different machines in the grid. The jobs may have specific dependencies that may prevent them from executing in parallel in all cases.

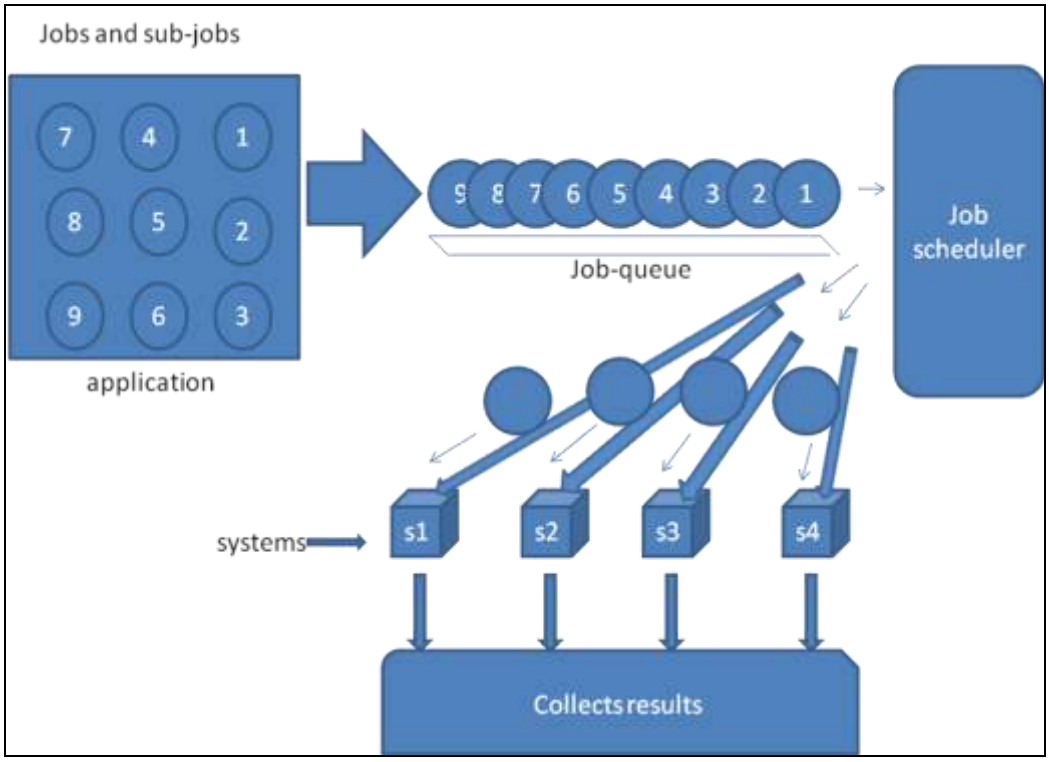

Fig. 1.1 An application is one or more jobs scheduled to run on grid

1.3.5 SCHEDULING, RESERVATION, AND SCAVENGING[19]

Advanced grid systems would include a job "scheduler" of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Sometimes the term "resource broker" is used in place of scheduler. Schedulers usually react to the immediate grid load. They use measurement information about the current utilization of machines to determine which ones are not busy before submitting a job. Schedulers can be organized in a hierarchy. For example, a meta-scheduler may submit a job to a cluster scheduler or other lower level scheduler rather than to an individual machine.

The term "scheduling" is not to be confused with "reservation" of resources in advance to improve the quality of service. Grid resources can be "reserved" in advance for a designated set of jobs.

In a "scavenging" grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job which is satisfied by the machine's resources.

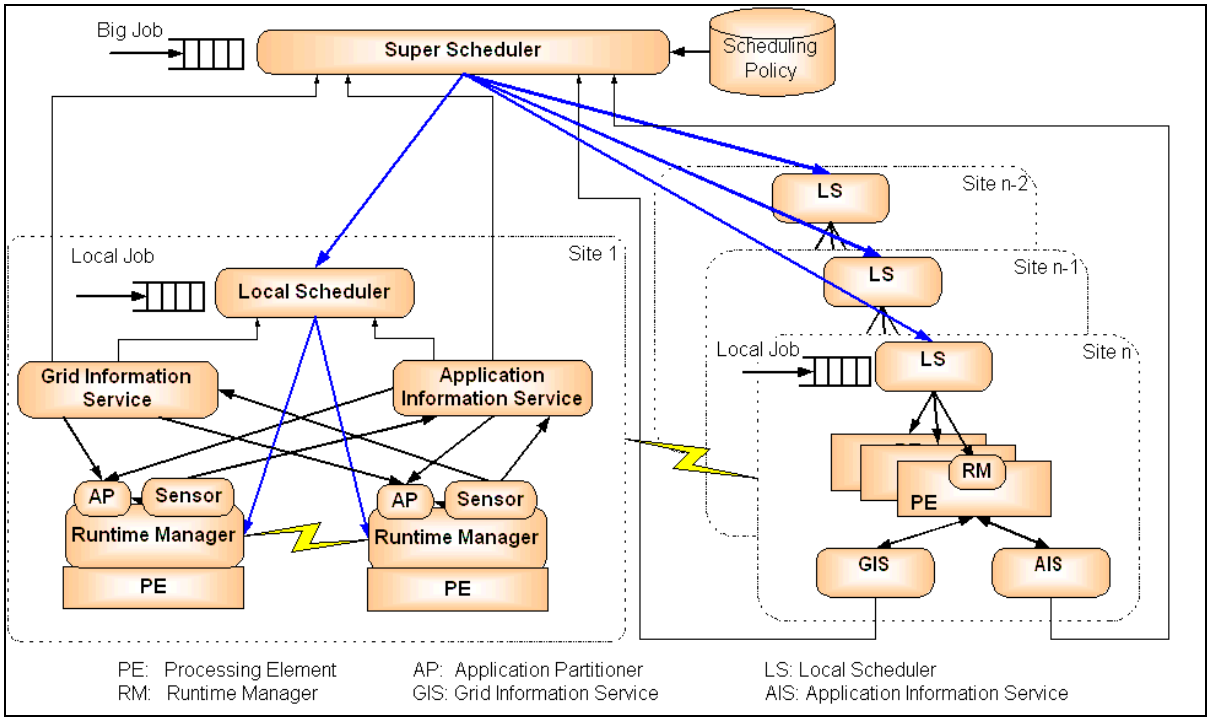

Fig. 1.2 Meta-scheduling in grid

1.3.6 INTRAGRID TO INTERGRID

The concept of grid computing is still evolving and most attempts to define it precisely end up excluding implementations that many would consider to be grids. Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as a hierarchy spanning the world. the simplest grid consists of just a few machines, all of the same hardware architecture and same operating system, connected on a local network. This kind of grid uses homogeneous systems so there are fewer considerations and may be used just for experimenting with grid software. The machines are usually in one department of an organization, and their use as a grid may not require any special policies or security concerns. Because the machines have the same architecture and operating system, choosing application software for these machines is usually simple. Some people would call this a “cluster” implementation rather than a “grid.”. Machines participating in the grid may include ones from multiple departments but within the same organization. Such a grid is also referred to as an “intragrid.” As the grid expands to many departments, policies may be required for how the grid should be used. grid may grow to cross organization boundaries, and may be used to collaborate on projects of common interest. This is known as an “intergrid.”

1.4 OBJECTIVE

The objectives of this thesis are to

- Identification of core grid functionalities and services for making operating system grid enable.
- Develop a modular architecture design for grid enabled operating system.

1.5 ORGANIZATION OF WORK

The rest of this thesis is organized as follows. Chapter 2 presents an overview of grid technologies, discusses some of the key software components of a grid, and presents a survey of representative systems. Chapter 3 presents the information of Linux kernel the targeted operating system because of its open source characteristic. Chapter 4 presents in-depth discussion of the architecture, its setup procedure and modular functionalities. Chapter 5 figures out the importance of proposed dgridOS architecture. Chapter 6 covers related topics to the dissertation work. Chapter 7 presents the conclusion and future direction of this work.

2. GRID ORGANIZATIONS

2.1 GRID INFRASTRUCTURES

An infrastructure is an underlying foundation that provides basic facilities, services and installations needed for the functioning of an organization or system. For example the Internet allows people to communicate with each other and virtually any electronic device, etc. In order to be widely deployed, an infrastructure must be simple and offer value to its users. Grid computing is an emerging infrastructure that provides scalable and secure mechanisms for the access, sharing and discovery of resources amongst dynamic collections of individuals and institutions. One of its goals is to provide these services in a manner that hides the implementation details from the user. The availability of high-speed networks, low cost components and the ubiquity of the Internet and Web technologies make it feasible to realize this vision. Grid Computing organizations and their roles can be broadly classified into four categories based on their functional role in Grid Computing. These roles are best described as:

- Organizations developing grid standards and best practices guidelines.
- Organizations developing Grid Computing toolkits, frameworks, and middleware solutions.
- Organizations building and using grid-based solutions to solve their computing, data and network requirements.
- Organization working to adopt grid concepts into commercial products, via utility computing, and Business on Demand computing.

2.2 GRID COMPUTING PROJECTS

A number of grid computing projects have undertaken the task of developing a grid infrastructure. Since grid computing is an evolving field, no standards as of yet have been developed and widely accepted. As such, many of these projects differ in their

implementation of the grid protocols and services discussed in Section. This section provides an overview of these projects.

2.2.1 GLOBUS TOOLKIT

Globus[4] is a grid-oriented community committed to developing standards for the protocols, APIs, services definitions and service behaviors associated with grid infrastructures. The community provides a toolkit that implements the basic components and services required to construct and support a computational grid. The infrastructure allows both users and applications working within the environment to view distributed heterogeneous resources as if they were local resources.

The Globus toolkit is constructed using a layered architecture that allows high level services to be built using low level services. The four main components of this architecture are resource management, data management, information services, and security. This architecture is shown in Figure.

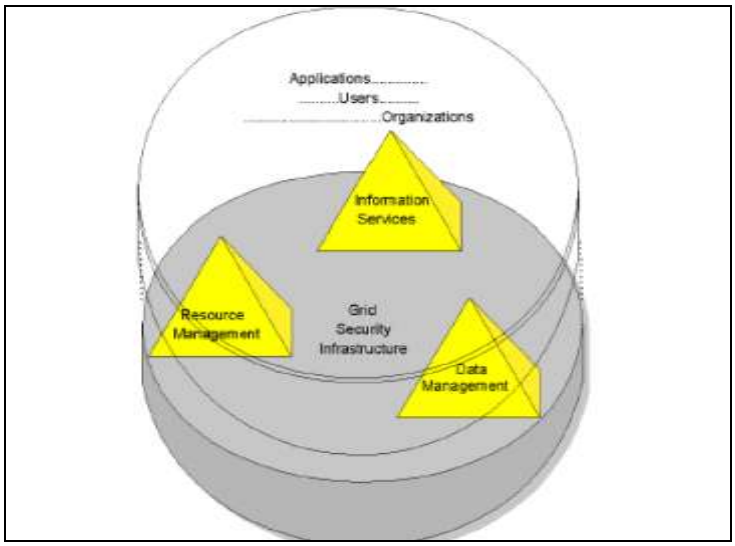

Fig. 2.1 Globus Toolkit Layered Architecture

The Grid Security Infrastructure (GSI) deals with the issues of authentication and authorization. Three important properties that GSI provides are single sign-on, mapping to local security mechanisms, and delegation. Single sign-on means that users only need to be authenticated once. When a user signs in, GSI uses public key encryption

mechanisms to verify the user's grid credential. The grid credential is a global credential used to authenticate the user to the grid infrastructure. GSI handles the mapping of grid credentials to local credentials in order to make use of authentication and authorization mechanisms in place on local systems. Delegation is provided by creating a proxy credential which is used for making requests on behalf of the user.

Information services handle the task of collecting information about resources in the grid and responding to queries about this information. This functionality is implemented in a component called the Monitoring and Directory System (MDS).A hierarchical directory system based on the Lightweight Directory Access Protocol (LDAP). MDS is further broken down into the Grid Resource Information Service (GRIS) and the Grid Index Information Service (GIIS). GRIS provides a uniform interface for querying a resource provider about the status of its resources and its configuration, while GIIS aggregate information into a single directory listing. A GRIS registers itself with a GIIS, and a GIIS can register itself with another GIIS, thus forming a hierarchy.

Resource management services handle the tasks of job submission, job execution management, and resource allocation. This functionality is implemented in a variety of different components. Grid Resource Allocation and Management (GRAM) is a protocol that supports the remote submission of a computational request to a remote computational resource, and the subsequent monitoring and control of the resulting computation. A command-line tool is used to submit job requests, transfer the job's executable file to the remote site, and transfer the resulting output files to the originating system. Requests sent by this tool are authenticated by a gatekeeper service, which starts a job manager service for each job submitted to the remote site. The job manager service handles job execution and job management operations. In addition, the gatekeeper is responsible for secure communication. Globus Access to Secondary Storage (GASS) is a service that provides GRAM mechanisms for transferring and accessing data between remote resources. A typical use of GASS is to transfer a job's executable and any of its inputs to a remote resource and transfer the resulting output back to the client. Globus resource management supports co-allocation through its Dynamically-Updated Request Online Co-allocator (DUROC) mechanism. DUROC allows jobs to be submitted to resources managed by different resource managers and coordinates the transactions between these managers.

Data management services handle file transfers and file transfer management. This functionality is implemented in GridFTP, a secure and reliable grid data transfer protocol based on the File Transfer Protocol (FTP). The Globus toolkit provides an implementation of both the GridFTP client and the GridFTP server.

A more detailed architecture of the Globus grid infrastructure is presented in Figure

Although Globus provides a fairly robust framework, problems with scalability can arise on the machine running the gatekeeper process when a large number of jobs are submitted to it. This is because an instance of a job manager is created for each job submitted. Once created, a job manager runs a programming script that continually probes the local scheduler about the status of a job at short intervals. This causes load average on the host machine to skyrocket, causing heavy swapping, and results in the bombardment of the local scheduler with potentially thousands of job status requests per minute.

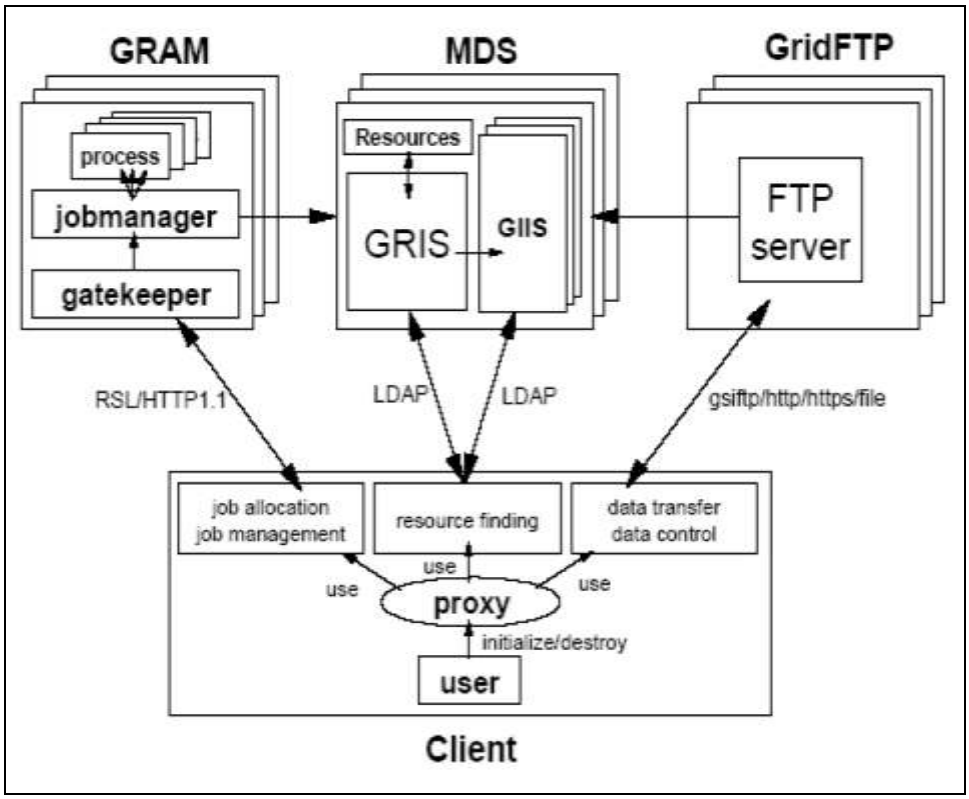

Fig. 2.2 Globus Toolkit Component Architecture

2.2.2 CONDOR

The Condor[9] project, from the University of Wisconsin, is aimed at investigating and developing mechanisms and policies that support high throughput computing on sets of distributed resources. The Condor system supports both dedicated and scavenged resources. Resources are discovered through centralized queries to a Condor pool manager. The Condor system uses a centralized scheduler. Resource requests and offers are expressed using Classified Advertisements (ClassAds), a semi-structured language which maps attribute names to expressions. Resources are represented by Resource-Owner Agents and customers are represented by Customer Agents. Resource-Owner Agents are responsible for enforcing resource usage policies put in place by resource owners and maintaining information about resource status. For example, a Resource-Owner agent might only accept applications for execution during specified time intervals. Both Resource-Owner Agents and Customer Agents submit their ClassAds to a ClassAds pool manager. The manager periodically invokes a matchmaking algorithm that scans submitted ClassAds and creates pairs that satisfy each others constrains and preferences. The matchmaking algorithm uses information about past resource usage policy in order to ensure an element of fairness. When a match is found, the pool manager "introduces" the matched parties to each other. If the Resource-Owner Agent accepts and runs the job submitted by the Customer Agent, it is said to be "claimed". Once the Customer Agent is finished with the resource, it releases its claim on it and the Resource-Owner Agent can advertise itself as unclaimed. It is possible to have a Resource-Owner Agent continue to advertise itself while it is claimed in order to listen for higher-priority jobs. The matchmaking process is analogous to scheduling.

Once a job is placed in an execution environment, it can run into a multitude of problems relating to missing libraries or files, firewalls, missing credentials, etc. One system might be aware of the user but not of the files the user needs and vise-versa. Only the execution system is aware of what file systems, networks, and databases must be accessed and how they can be reached. Condor uses split execution in order to address these issues. Split execution involves the cooperation of a shadow, responsible for specifying everything the job needs at run time, and a sandbox, responsible for providing the job with a safe execution environment. The combination of a shadow and a sandbox is referred to as a universe. Universes are used in order to preserve a job's originating environment on the

execution machine. The Condor system was extended in order to be able to submit jobs to grid resources using services provided by the Globus toolkit. This extended framework is called Condor-G. The main component of this system is the Condor-G agent, which allows a user to treat the grid as a local resource. It provides an API and command-line tools that allow the user to submit jobs, cancel jobs, obtain job-related information and perform grid queries. Once the agent receives a job submitted by the user, it stages the job's standard input/output (I/O) and executable using Globus GASS, submits the job to a remote resource using Globus GRAM, handles job monitoring and failure recovery through Globus GRAM and authenticates all requests via Globus GSI mechanisms. Condor's implementation of the GRAM protocol is revised in order to make it more fault-tolerant. It uses a two-phased commit protocol during job submission in order to deal with lost requests and responses and logs the details of all active jobs to stable storage at the client side in the event of local failure. The Condor-G agent also handles communication with the user concerning errors or unusual conditions, resubmission of failed jobs, and stores state information for each submitted job to persistent storage in the scheduler's job queue to support restart in case of failure. The system architecture is presented in Figure

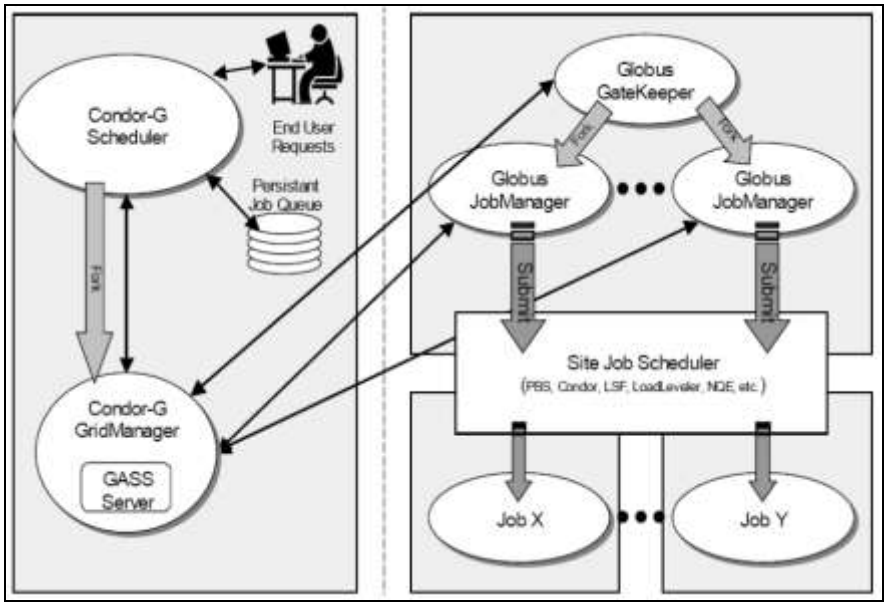

Fig. 2.3 Condor-G architecture

2.3.3 LEGION

Legion[1] is an object oriented meta-computing system designed to consolidate and manage large collections of heterogeneous, distributed machines and present the

users with a single coherent view of this system. Legion does not mandate any particular programming model or language; it only defines APIs for object interaction. The Legion framework defines five main class objects which represent components of a grid. They are a Host, a Vault, a Collection, a Scheduler, and an Enactor. Class objects have two main purposes in Legion: Defining types for their instances and acting as managers for these instances. Host objects represent the machines shared as part of the grid. Host objects implement functions for load monitoring, reservation management, and information retrieval. Host objects assign attributes with information about their current state, architecture, operating system, etc. Vault objects are used in order to store persistent information regarding Host objects. Collection objects represent a repository of state-related resource information. It allows system administrators to organize resources into suitable arrangements, such as an organizational topology based on the location of machines. Collections may retrieve information directly from resources or may share information with other Collections. Collections only supported static resource information. Scheduler objects obtain resource information from Collections and use this information to compute mappings from objects to resources. Once these mappings are determined, they are passed to an Enactor object for verification and instantiation. Each schedule consists of at least one master schedule, which might contain a list of alternate schedules. Enactor objects verify resource mappings passed to them by Scheduler objects and perform the instantiation of these mappings. It does this by verifying whether an object can be instantiated on the machine at the given time. If all of the mappings in the master schedule succeed, the scheduling is complete. Instantiation is held off until a Scheduler object calls a method that tells the Enactor object to perform the instantiation on the reserved resources. If any of the mappings fail, an alternate schedule for the failed mapping is selected and verified. Mappings can fail if the resources on which the object should be instantiated are unavailable. Interaction between objects in the Legion framework is demonstrated in Figure

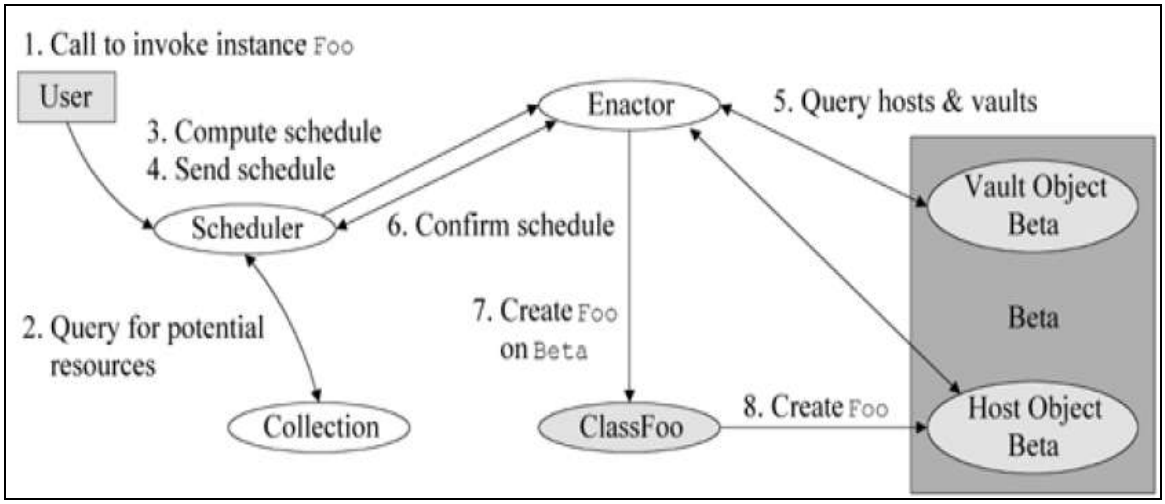

Fig. 2.4 Legion Component Interaction

Job handling in Legion is accomplished via a set of command-line tools that allow programs to register themselves with the Legion system and start program instances. The procedure for running a Legion-enabled program is fairly straightforward: the class associated with the program is instructed to create an instance of the class which executes the program code. Running a non-Legion enabled program involves a more complex procedure; a "BatchQueueClassObject" metaclass must be used in order to create a specialized class object which acts as the class object for the non-Legion enabled program. "JobProxyObjects" are used in order to manage the execution of a program. A JobProxyObject is created whenever the class associated with a program is instructed to create an instance of itself. There are several disadvantages to job handling in the Legion framework. One JobProxyObject is required for each running job, thus increasing the overall resource requirements for each job. Also, there is no method of monitoring or restarting a job, and there is no control over the total number of jobs executing simultaneously.

2.2.4 NIMROD

Nimrod[2] is a software system whose purpose is to manage the execution of parametric studies across distributed computers. Parametric studies execute one application many times with different sets of input parameters. Nimrod provides the facilities to create, execute, monitor and collect the results of individual experiments. Nimrod requires experiments to be described through the use of declarative plan files. These files contain the parameters and the commands necessary to perform the work.

Nimrod uses this information in order to transfer the necessary files and schedule the work on the first available machine. The plan file is processed by a generator tool, which allows the user to choose values for the parameters specified in the file. Once this is done, the generator builds a run file that is processed by a dispatcher tool. The dispatcher is responsible for managing the computation across the nodes on which the computation is scheduled to run. Nimrod does not operate well in a grid environment due to several limitations. It operates on a static set of resources, it has no provision for dynamic resource discovery, it does not understand the concept of user deadlines, and it has no support for a variety of access mechanisms. Nimrod-G is an extended version of Nimrod which is designed to run in grid environments. It takes advantage of grid resources and services and uses a model of computational economy as part of the Nimrod-G scheduler.

The Nimrod-G architecture consists of five main components: a client/user, a parametric engine, a scheduler, a dispatcher and a job wrapper. The architecture is very flexible and can be integrated with grid services provided by systems such as Globus, Legion and Condor. The architecture is shown in Figure

The client is an interface for controlling and supervising an experiment under construction. It allows users to modify parameters related to time and cost, which influences how the scheduler selects the resources on which to run the experiment. The client also allows the user to control, monitor and check the status of jobs.

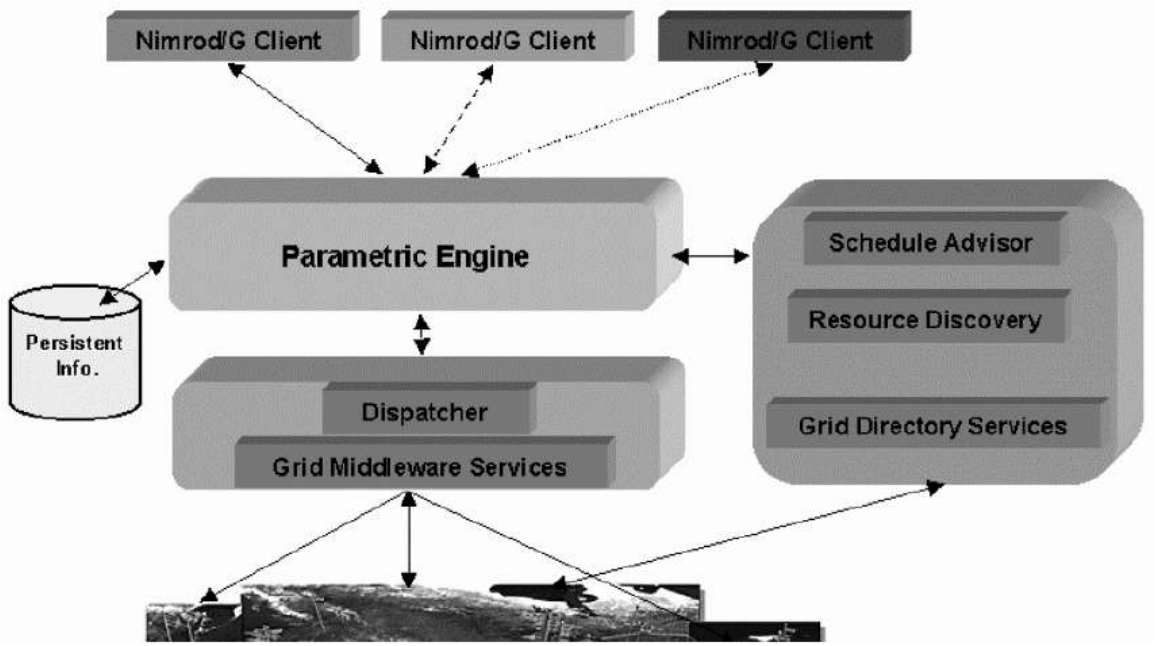

Fig. 2.5 Nimrod-G architecture

It is not bound to any particular site; the user can shut it down and start it on another system without affecting the outcome of the experiment. It also is possible to have many clients connected to and controlling or monitoring the same experiment. The parametric engine acts as a central coordinator for the experiment. It is responsible for parameterization of the experiment and the actual creation of jobs, maintenance of job status, interacting with clients, schedule advisor, and dispatcher. The engine also ensures that the state of the experiment is stored in persistent storage in the event of failure. The scheduler is responsible for resource discovery, resource selection, and job assignment. It uses a resource selection algorithm based on a model of computational economy, which selects resources according to those that meet the deadline and minimize the cost associated with a computation. In the basic Nimrod model, once tasks are started they do not communicate with one another; scheduling is simply a problem of finding suitable resources and executing the experiment. The dispatcher is responsible for initiating the execution of a task on a selected resource and updating its status to the parametric engine. This is accomplished by starting a job wrapper on the selected resource. The job wrapper is responsible for staging the tasks and data associated with an experiment, executing the tasks and sending the results back to the parametric engine via the dispatcher. Resource selection can be handled in two different ways when using a model of computational economy. One method is to allow the system to work on behalf of the user and attempt to complete the assigned work within a given deadline and cost. Another method is to allow the user to enter into a contract with the system. The contract specifies what the users are willing to pay for the resources if the work can be completed within a given deadline. Using this method, the system can identify a set of viable resources through the use of resource reservation or trading mechanisms. If the user is satisfied with the contract, it can be accepted. Otherwise, the contract can be re-negotiated by changing the deadline and/or cost constraints. The latter method is advantageous since it allows the user to be aware of whether the work can be completed within the given deadline and cost constraints before the work actually starts. However it requires grid middleware services for resource reservation, broker services for cost negotiation, and an underlying system which has a management and accounting infrastructure. Nimrod-G uses the Grid Architecture for Computational Economy (GRACE), an economic framework which provides the services that helps resource providers and resource consumers maximize

their goals. Resource providers can use GRACE mechanisms to define their charging and access policies and the GRACE trader works according to these policies. Resource consumers define their requirements through the use of resource brokers, who use grace services for resource trading and identifying resource providers that meet their needs. The GRACE framework is generic enough to accommodate different economic models.

3. INTRODUCING LINUX KERNEL

3.1 INTRODUCTION

Linux[8] is a member of the large family of Unix-like operating systems. Linux was initially developed by Linus Torvalds in 1991 as an operating system for IBM-compatible personal computers based on the Intel 80386 microprocessor. One of the more appealing benefits to Linux is that it isn't a commercial operating system: its source code under the GNU Public License is open and available to anyone to study. If we download the code (the official site is http://www.kernel.org) or check the sources on a Linux CD, we will be able to explore, from top to bottom, one of the most successful, modern operating systems. Because of its openness it has been selected for this demonstration. Most Unix kernels[8] are monolithic: each kernel layer is integrated into the whole kernel program and runs in Kernel Mode on behalf of the current process. In contrast, *microkernel* operating systems demand a very small set of functions from the kernel, generally including a few synchronization primitives, a simple scheduler, and an inter-process communication mechanism. Several system processes that run on top of the microkernel implement other operating system-layer functions, like memory allocators, device drivers, and system call handlers.

The main advantages of using modules include:

A modularized approach: Since any module can be linked and unlinked at runtime, system programmers must introduce well-defined software interfaces to access the data structures handled by modules. This makes it easy to develop new modules.

Platform independence: Even if it may rely on some specific hardware features, a module doesn't depend on a fixed hardware platform. For example, a disk driver module that relies on the SCSI standard works as well on an IBM-compatible PC as it does on Hewlett-Packard's Alpha.

Frugal main memory usage: A module can be linked to the running kernel when its functionality is required and unlinked when it is no longer useful. This mechanism also can be made transparent to the user, since linking and unlinking can be performed automatically by the kernel.

No performance penalty: Once linked in, the object code of a module is equivalent to the object code of the statically linked kernel. Therefore, no explicit message passing is required when the functions of the module are invoked.
Technically speaking, the operating system is considered as the parts of the system responsible for basic use and administration. This includes the kernel and device drivers, boot loader, command shell or other user interface, and basic file and system utilities. The term system, in turn, refers to the operating system and all the applications running on top of it.

The user interface is the outermost portion of the operating system, the kernel is the innermost. It is the core internals. The software that provides basic services for all other parts of the system, manages hardware, and distributes system resources. The kernel is sometimes referred to as the supervisor, core, or internals of the operating system. Typical components of a kernel are interrupt handlers to service interrupt requests, a scheduler to share processor time among multiple processes, a memory management system to manage process address spaces, and system services such as networking and inter-process communication. On modern systems with protected memory management units, the kernel typically resides in an elevated system state compared to normal user applications. This includes a protected memory space and full access to the hardware. This system state and memory space is collectively referred to as kernel-space. Conversely, user applications execute in user-space. The see a subset of the machine's available resources and are unable to perform certain system function, directly access hardware, or otherwise misbehave (without consequences, such as their death, anyhow). When executing the kernel, the system is in kernel-space executing in kernel mode, as opposed to normal user execution in user-space executing in user mode. Applications running on the system communicate with the kernel via system calls. An application typically calls functions in a library for example, the C library that in turn rely on the system call interface to instruct the kernel to carry out tasks on their behalf. Some library calls provide may features not found in the system call and thus, calling into the kernel is just one step in an otherwise large function. For example, consider the familiar printf() function. It provides formatting and buffering of the data and only eventually calls write() to write the data to the console. Conversely, some library calls have a one-to-one relationship with kernel. Fro example,

the open() library function does noting except call the open() system call. Still other C library functions, such strcpy(), should make no user of the kernel at all. When an application executes a system call, it is said that the kernel is executing on behalf of the application. Furthermore, the application is said to be executing a system call in kernel-space, and kernel is running in process context. This relationship that applications call into the kernel via the system call interface is the fundamental manner in which applications get work done.

The kernel also manages the system's hardware. Nearly all architectures, including all systems that Linux supports, provide the concept of interrupts. When hardware wants to communicate with system, it issues an interrupt that asynchronously interrupts the kernel. Interrupts are identified by a number. The kernel uses the number to execute a specific interrupt handler to process and respond to the interrupt. For example, as you type, the keyboard controller issues an interrupt to let the system know that there is new data in the correct interrupt handler. The interrupt handler processes the keyboard data and lets the keyboard controller know it is ready for more data. To provide synchronization, the kernel can usually disable interrupts either all interrupts or just one specific interrupt number. In many operating system, including Linux, the interrupt handlers do not run in a process context. Instead, they run in special interrupt context that is not associated with any process. This special context exists solely to let an interrupt handler quickly respond to an interrupt, and then exit.

These contexts represent the breadth of the kernel's activities. In fact, in Linux, it is generalized that each processor is doing one of three things at any given moment:

- In kernel-space, in process context, executing on behalf of a specific process
- In kernel-space, in interrupt context, not associated with a process, handling an interrupt
- In user-space, executing user code in a process.

This list is inclusive. Each corner cases fit into one of these three activities: For example, when idle, it turns out that the kernel is executing an idle process in process context in the kernel.

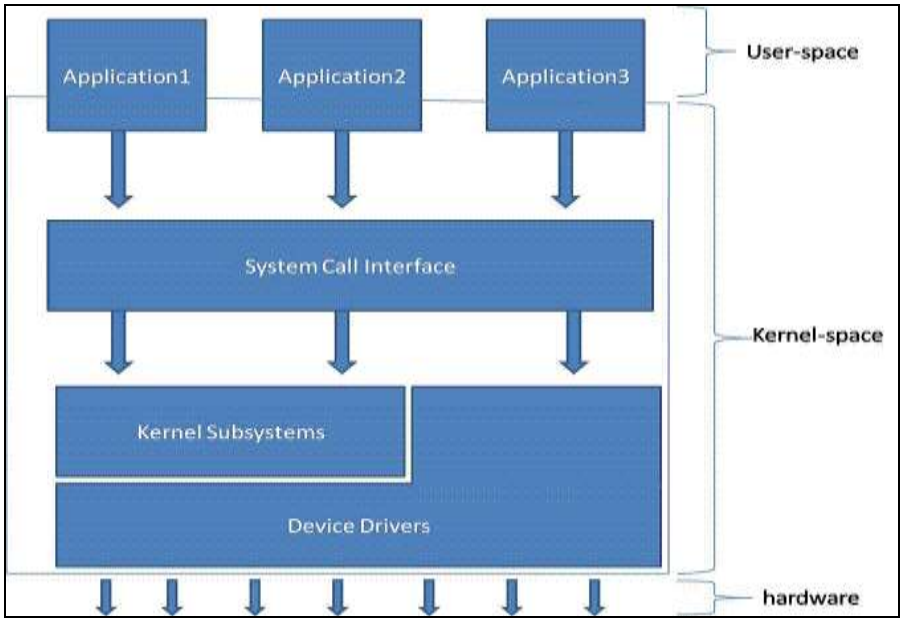

Fig. 3.1 Relationship between applications, the kernel, and hardware

3.2 SCOPE FOR GRID MODULES

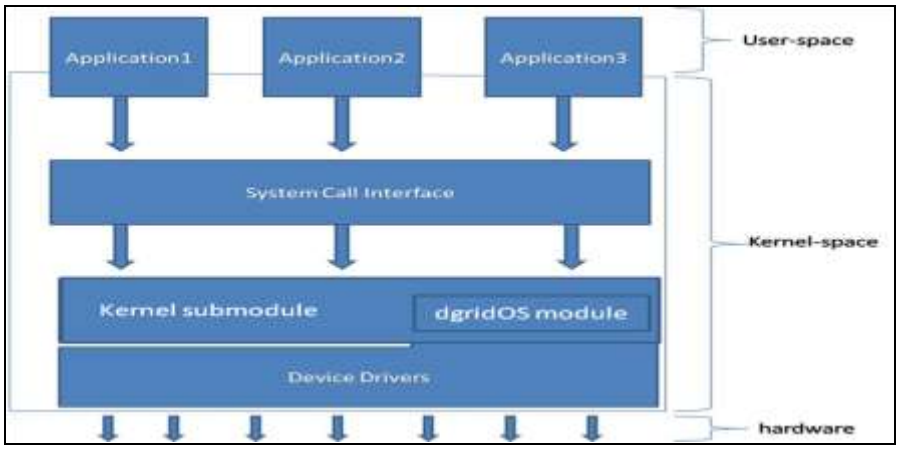

Fig. 3.2 Grid services module installed inside the kernel-space

As shown in the Fig.3.2 the grid services module can be integrated inside the kernel-space without necessarily disturbing other modules and architecture. Since microkernel approach of operating system supports module-integration and de-integration at runtime. It can be easily bundled to kernel-space.

4. DESIGN OF OPERATING SYSTEM WITH GRID ENABLED SERVICES

4.1 INDENTIFICATION OF CORE SERVICES[11]

- For a single node
  - Resource discoverer and Characterizer
  - Resource publisher
  - File system identifier and virtualizer
  - Security Controller
  - Kernel Cache to store information and its Controller
  - Job executor and Process controller
- For node inside a gridLan
  - Communication Controller
  - Informer and Updater to gridLan head node.
  - Node identifier
  - Cache synchronizer
  - process synchronizer
  - Virtual file system manager
- For a gridLan
  - Resource Broker
  - Resource Scheduler

4.2 PROPOSED GRID ARCHITECTURE

Current grid implementation includes grid middleware software which is responsible for formulation and support for grid computing. It provides services to application developers so that they can write code, which can exploit underutilized resources. Middleware is responsible for binding the resources into a single virtual computing resource.

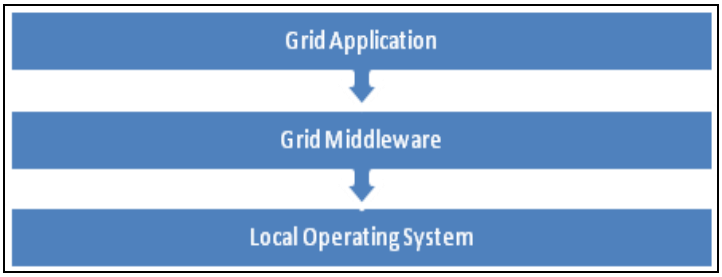

Fig. 4.1 Current Grid implementation

Our grid implementation eliminates the grid middleware and the supporting functionality is directly included inside the operating system. Applications no longer need to be middleware specific in order to use its functionality.

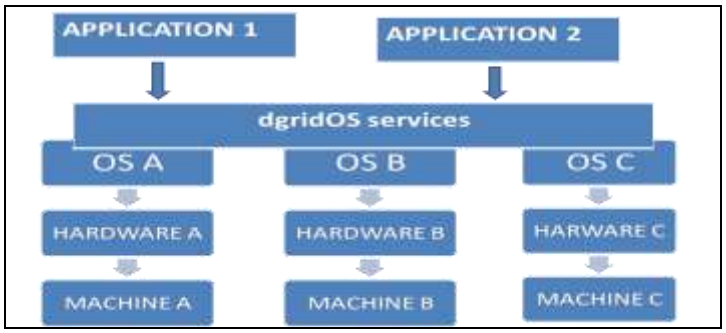

Fig. 4.2 proposed grid implementation

The proposed architecture reveals that applications are submitted to dgridOS without any intermediate middleware tool or services. Applications are straight forward served by dgridOS functionalities which actually binds all the hardware and software resource into single virtual resource pool. dgridOS services do almost all the functions provided by previous middleware.

4.3 VIRTUALIZATION CONCEPT

The Fig. 4.3 below describes the virtualization concept. The sub modules used in the graph are dgridOSvfs: dgridOS virtual file system module and dgridOSres: dgridOS resource module.

dgridOSvfs module maps the node file system to the common dgridOS file system and attached it to the central dgridOS headnode virtualized tree like structure.

dgridOSres module, collects all the parameter values of its different types of resources into a structure and publish it to head node of dgridLan. The module dgridOSres of concerned node and head node communicate and manage the overall availability of resources.

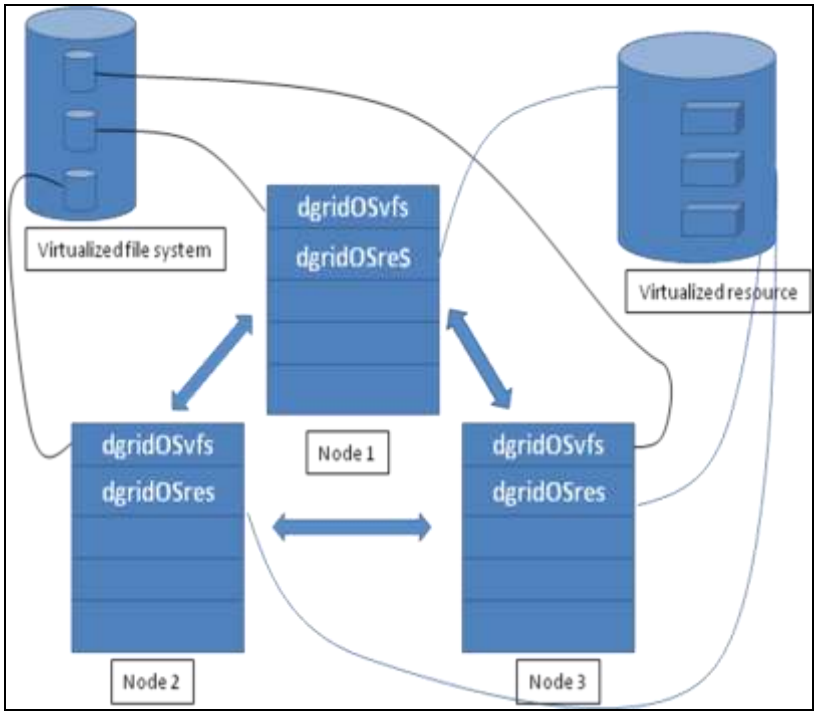

Fig. 4.3 Virtualization concept on dgridOS architecture

4.4 TOPOLOGY FOR dgridOS IMPLEMENTATION

Identifications

The proposed operating system is named as "dgridOS". It works on a topology with group of computers inside a LAN forms a "gridLan", a small grid with one head note and all others as participation nodes. A node can be simple desktop computer with computing power, memory and other resources. A gridLan can be virtualized through the concept of a cluster but is not designed for clustering services. The head nodes of gridLan have additional functionality to integrate them selves to form a wide structure of overall grid.

.

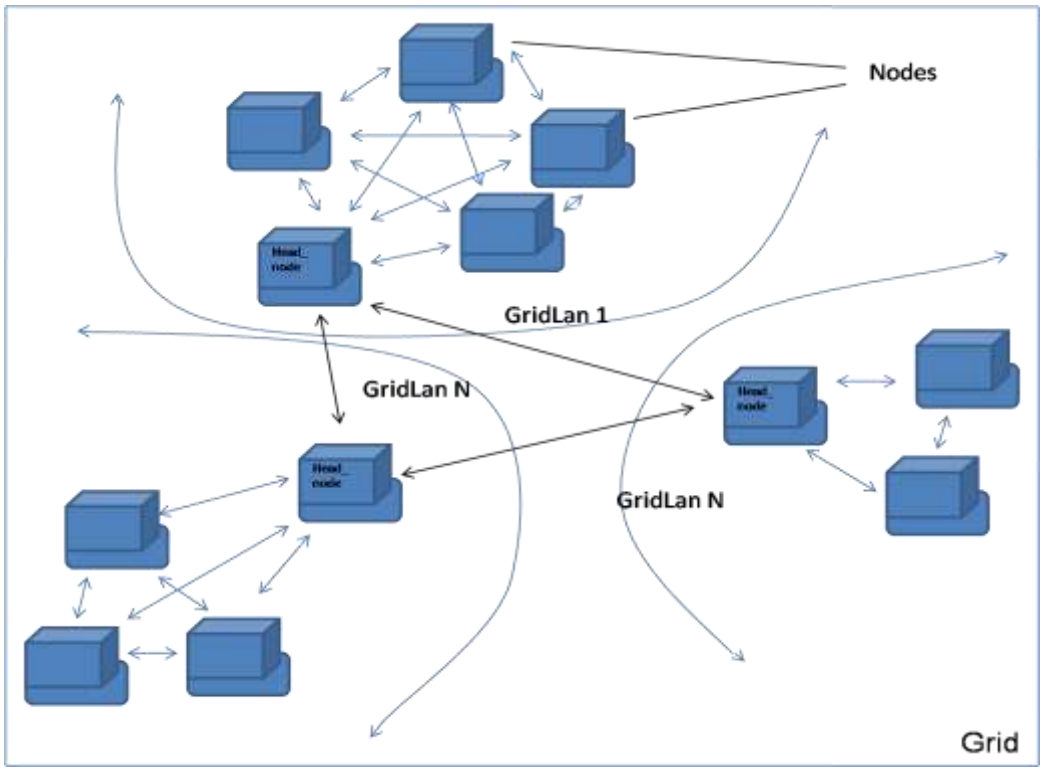

Fig. 4.4 dgridOS architecture topology

4.5 MODULAR ARCHITECTURE OF dgridOS

The modular architecture of dgridOS includes five core modules which gets instantiated through the dgridOScore.

1. dgridOScom module: This is basically dgridOS communication control module.
    a. This module maintains its gridLan buffer, where it keep updating the listening nodes inside its gridLan.
    b. It broadcast its address, and update its buffer in accordance with the head node girdLan buffer.
    c. This module keeps records of ports and services offered on the ports.
    d. A peer to peer network is established on any particular node (choose from the free pool of non standard ports) for communication among nodes of gridLan.
2. dgridOSsec module: This is dgridOS security control module.
    a. This module checks for the user authentication, authorization.
    b. It maintains any restricted resource or service.

   c. If authentication fails then it stops communication to that node and accordingly publish it to head node.
   d. It manages log of user login, logout records.
3. dgridOSres module: This is dgridOS resource control module.
   a. It includes a cache of its resource descriptions.
   b. It sends informations of its available resources to head node dgridOSres module.
   c. A part of it, works for resource virtualization with head node inside a gridLan and accordingly part in resource brokering and scheduling.
4. dgridOSvfs module: This is dgridOS virtual file system control module.
   a. It starts with the recognition of node file system.
   b. If its type is known, it is mapped with the dgridOS file system.
   c. It is linked to the central hierarchy of virtual file system and maintained.
5. dgridOSpro module: This is dgridOS process control module.
   a. It includes jobCache in order to track process/job proceedings.
   b. A part of it, manages the process related communication with head node's process control module.
   c. A part of it, manages to execute processes in an optimized way using the virtual resource.
   d. A part of it, manages inter-process communication and synchronizations.

Apart from these, the modular architecture has brokering and scheduling module which works over the framework buildup by dgridOScore.

This modular structure is placed inside the kernel space of operating system.

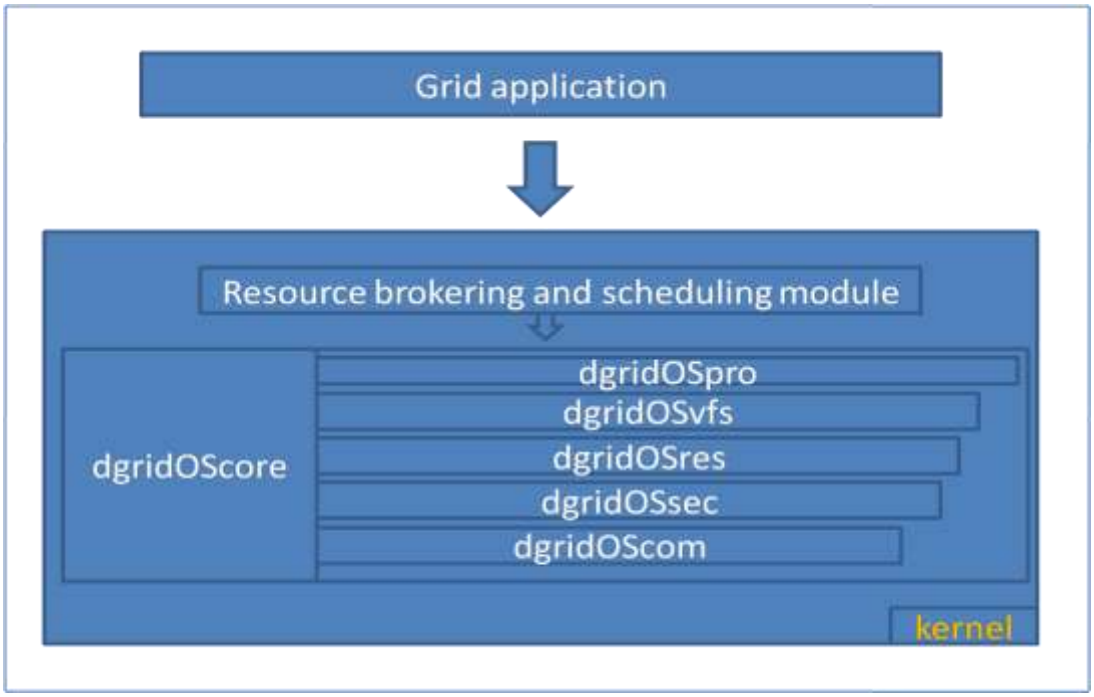

Fig. 4.5 dgridOS architecture

INTEGRAL PARTS OF MODULES ARE DESCRIBED IN THE FOLLOWING SECTIONS.

(From the side of Client (or participating nodes) inside a gridLan)

dgridOScore: This module is the heart of dgridOS, calls all other dgridOS modules inside its definition.

dgrid_Initializer()	• Sets the values of dgrid environment variables. • Creates dgridLocalCacheBuffer • Collects local resources (computational, primary memory, disk memory,communicational) parameter values using kernel and device driver services. • Sets security mode ON • Stores local resources parameters, security mode and other variables values in a structured database of dgridLocalCacheBuffer.
dgrid_bufferMgr()	• Checks for any modifications on dgridLocalBufferCache (since it is a long term buffer, need no updation till, hardware or notified software gets changed). • In case of hardware/ concerned software update, it does 1. insertion

	2. deletion 3. updation
dgrid_Launcher()	• Launches serially dgridOScom module, dgridOSsec module, dgridOSres module, dgridOSvfs module, dgridOSpro module.
dgrid_StartupChecker()	• It queries for checking the parameters of modules. • It maintains a log for failure of modules check and accordingly provide alert.

Table 4.1 Integral parts of dgridOS core module

dgridOScom: This is basically communication establishment module. Protocols related to communication can be used or can be newly created for node discoveries and development of networks.

dgrid_publishIP()	This command broadcast node ip in local network.
	• uses parameters:
	  • String ipaddress
	  • String subnetmask (of gridLan)
	  • String netmask (used in case of inter gridLan communication)

dgrid_addrCollector()	It uses the positive responses of dgrid_publishIP command and update them in dgrid_gridLanBuffer.
	• structure fields are
	  • int node_id (unique inside the gridLan assigned by head_node)
	  • String ipaddress
	  • String subnetmask address
	  • String netmask address

dgrid_gridLanBufferMgr()	While dgrid_addrCollector collects the addresses and assign the unique id to nodes, dgrid_gridLanBufferMgr does the management of that buffer. • less frequently checking the validity of address to other nodes dgrid_comBuffer through their

	dgrid_comBufferMgr. • Updation [dgrid_addrCollector's Structure pointer] • Insertion[dgrid_addrCollector's Structure pointer] • Deletion[dgrid_addrCollector's Structure pointer]
dgrid_portsMgr()	This module picks up a port number (non standard port) and stablish a peer to peer network on that specified port inside a gridLan. • The dgrid head_node communicate on this specified port inside the gridLan and on a different assigned port outside the gridLan for interlinking to other dgrid heads. • It also relate different dgrid services to different port numbers. • The basic structure pattern include o String dgrid_gridLan_id. o String dgrid_gridLanassignedPortNumber list. o String* dgrid_servcie_portPointer [string dgridserviceName, int portNumber] (this is cached at every node inside gridLan in their dgrid_gridLanbuffer)
dgrid_comServiceMgr()	• int * getfreeports( ) • int periodicBandwithUtilization(hostdgrid_node_id, targetdgrid_node_id, comm._res pointer) [Checks the load on a communication channel, thus congestion can be controlled] • insert_services(dgrid_services_portNumber, node_id) • delete_services(dgrid_services_portNumber, node_id) • update_services(dgrid_services_portNumber, node_id)

Table 4.2 Integral parts of dgridOS communication module

dgridOSsec: This is the security control module. It checks the authentication, authorization users of partitioning nodes and resources.

dgrid_authenCltr()	Management of identity and verification of nodes to server and recognition of each other.
dgrid_privCltr()	Management and control of privileges to users/ peers.
dgrid_shareLimits()	Control and management of highly secure part away from shared pool.
dgrid_logCltr()	Logging on/off management and control
dgrid_authCltr()	Authorization management and control unit for dgrid.

Table 4.3 Integral parts of dgridOS security control module

dgridOSres: This is basically resource identification, integration and virtualization module. Resource characterization into computational, storage, communication, or other categories, Their respective parameterization are defined. Resource discovery, integration, deletion and status check functionality is provided by this module.

dgrid resource types	• Computation • Storage • Communication • Software and Licenses • Special equipments (like printer, scanner)

dgrid resource descriptions	It uses the type name from above modules and specifies the parameters corresponding the resource description structure of each type.
	
	Resources are indentified with the prefix of its node_id inside a gridLan.
	
	• structure descriptions
	  • Computational
	  • name
	  • MIPSrating (million instruction per second)
	  • costpersec (used in scheduling and brokering services)
	  • architecture [16bit, 32bit, 64bit, other]
	  • numberofresource (used in calculating overall computational cycles)
	  • Storage

	• type [primary]
	  • size (in megabytes)
	  • unallocated size
	  • allocated size
	• type [disk]
	  • size (in megabytes)
	  • unallocated size
	  • allocated size
	• Communicational
	  • name
	  • bandwidth
	  • peakLoad
	  • allocation policy [yes/no, type]
	  • currentLoad (used in order to avoid network congestion)
	• Software and Licenses
	  • this module is not included so far(considering the size of kernel).
	• Special equipment
	  • different integer values for printer, scanner, other devices.

dgrid_resCache	• It stores the statistics of resources in combining both of the structures defined by dgrid resource Type and dgrid resource description into a single structure. • This is maintained on its own node.
dgrid_resMgr()	This module coordinates with the head node resCache. • It maintains and manages the resources on its own node. • It sends resource request to head node in case of need and maintains the log. • It allocates resource on head node request and maintains the log. • It sends updates for resource allocation or deallocation to head node's dgrid_ resMgr().

Table 4.4 Integral parts of dgridOS resource managment module

dgridOSvfs: This module develop a virtual file system by vitualization of resources inside the grid and thus work on common file system.

dgrid_fsRecog()	• This gets the file system type • Maps with the grid known file system type(if

	known)
dgrid_vfsMgr()	defines management policies with working as on gridLan. Control the mount point statistics from dgridLan file system. And on process to this mount point it controls • UpdatedOnVfs() • WriteOnVfs() • ReadOnVfs() • DeleteOnVfs()

Table 4.5 Integral parts of dgridOS virtual file system module

dgridOSpro: This is basically process control block. Processes parallelization its scheduling and synchronization is maintained by this module.

dgrid_jobListener()	This module gets started with the dgridOScore module and starts listening for any grid job or application. This includes • startJobListener() 1. waitJobOccurrence() and 2. serveEvent() cycles • It serves the request • Goto step 1: waitJobOccurence()
dgrid_jobCache	In case of job has occurred, its statistics are maintained inside this module. structure for job is • int Job_id • int subJob_id • int computational_power_used • int memory_used • int status • boolean migrated[yes/no]). • allocated_resource_list • input cache • output cache
dgrid_processMgr()	Deals with migrated job processes and uses the routines • dgridJobResolver()

	• dgridFailoverResolver() • dgridJobContextUpdater()
dgrid_outputCollector()	It collects the output from output buffers. Uses the following routines • getdgridJobOutputReader() • getdgridJobId() • getdgridSubJobId() • releaseJobOutput()
dgrid_cleaner()	It cleans the records from buffer for the process which get successfully completed. • identifyJobsCompleted() • deleteJobsCompleted(dgrid_Job_id pointer)

Table 4.6 Integral parts of dgridOS process control module

Resource Brokering and Scheduling module:

This module runs over the abstract created by above mentioned modules. It works on combined pool of resources. It takes the application / jobs/ batch processes and efficiently distribute. . It dynamically find, identify, characterize, evaluate, allocate the most suitable resources to the user's application.

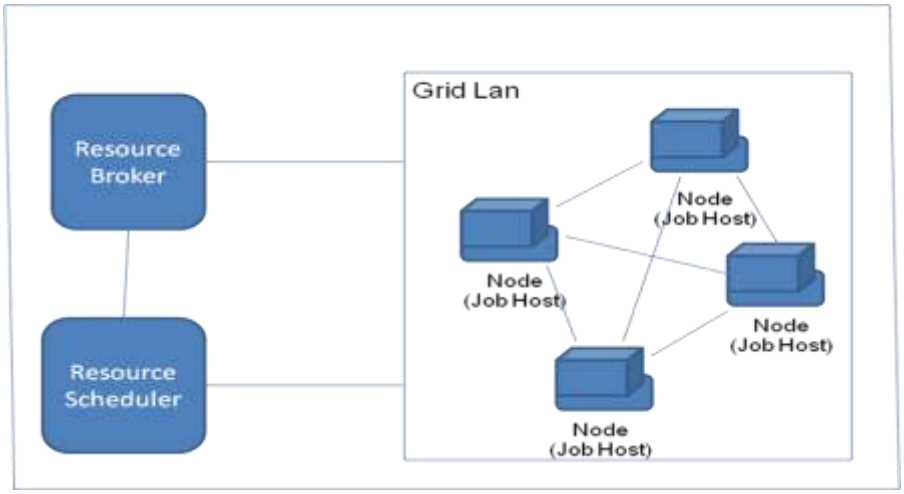

Fig. 4.6 dgridOS Resource brokering and Scheduling

4.3 JOB EXECUTION

dgridOS setup

Job execution is carried on the framework created by dgridOScore. dgridOS setup procedure does the creation of framework for grid application. It starts with dgridOScore module, which initializes the dgrid variables and caches, followed by the sequential call of other modules as described the following Fig. 4.5.

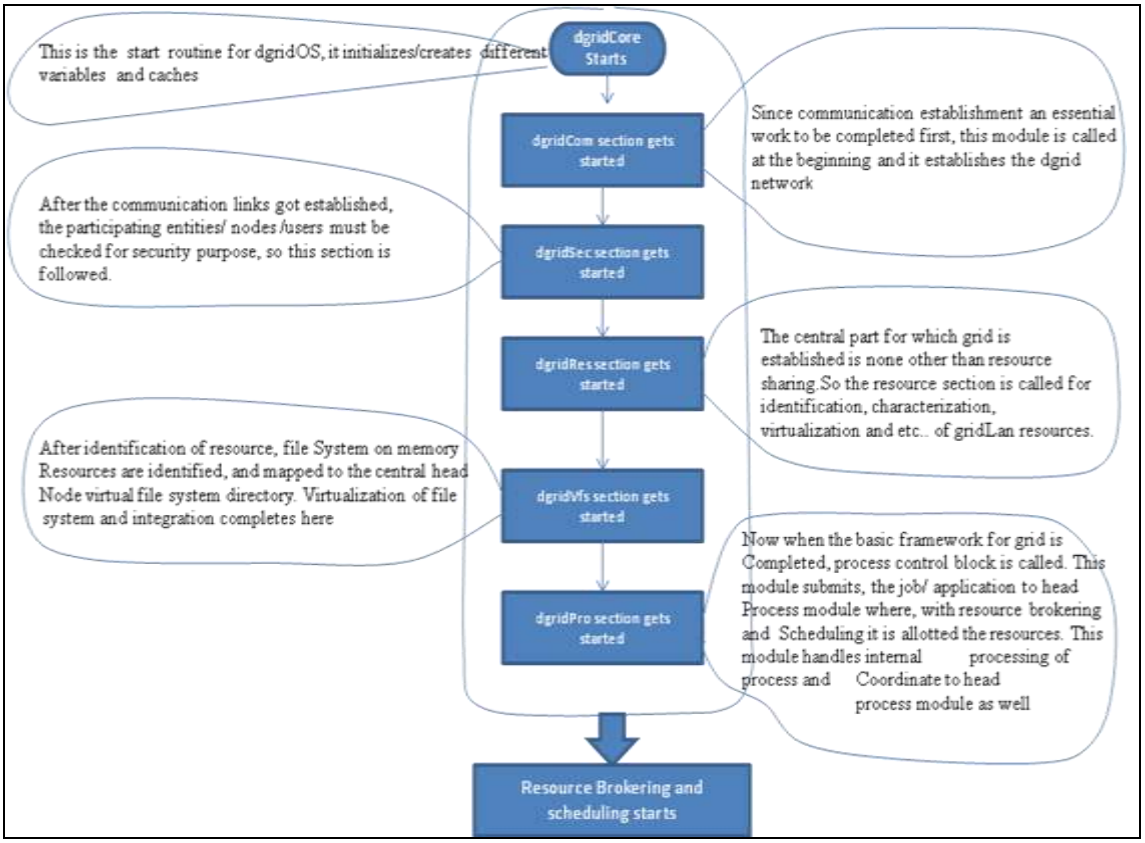

Fig. 4.7 dgridOS setup procedure

Job execution in dgridOS environment.

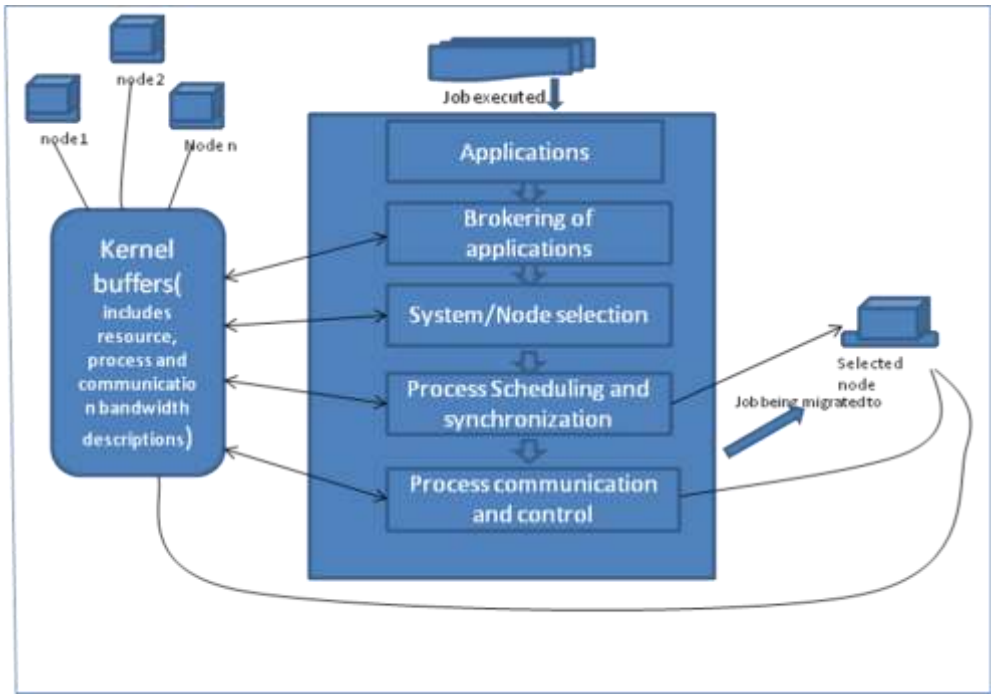

Fig. 4.8 job execution on dgridOS architecture

- Application is submitted to brokering module, where broker identifies the values of application characteristics.
- Based on the characteristics parameters of job, broker uses its metrics to find the optimal node where application should execute.
- Process control block get an application/ job and integrate it inside into scheduling and synchronization model.
- While the processes are executing inter process communication is monitored, and controlled and the output is collected to output cache inside the kernel buffer.

5. ADVANTAGES OF dgridOS

5.1 OPTIMIZATION IN FILE TRANSFER

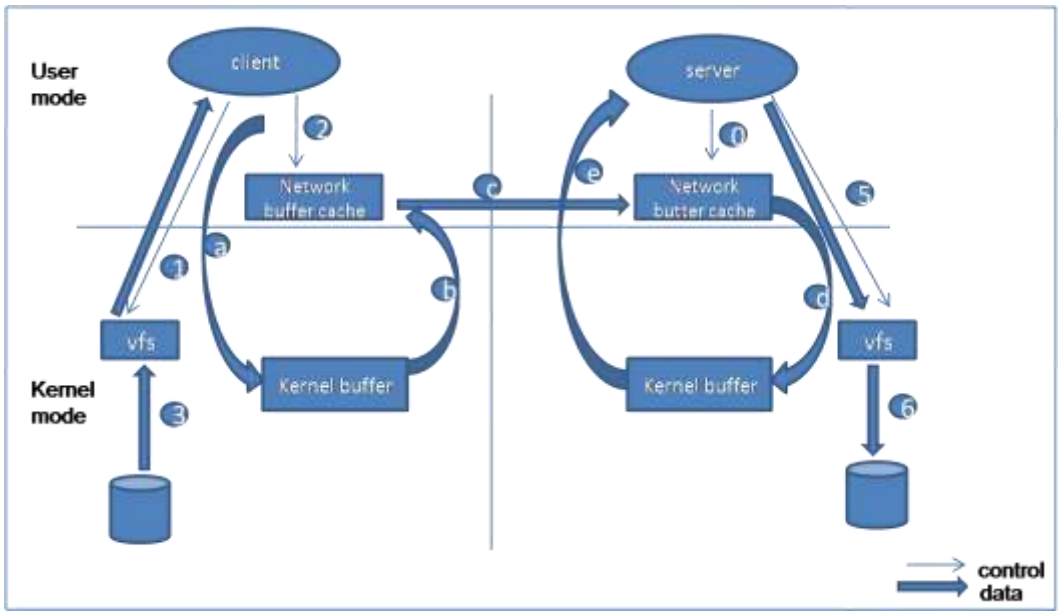

Fig. 5.1 File transfer from client to server without dgridOS services.

The steps a, b, c, d, and e of previous design consideration are eliminated in the grid enabled operating system. Since the network buffer is part of operating system kernel space, it does not need to communicated from user mode to kernel mode while transferring a file from client to server or vice versa.

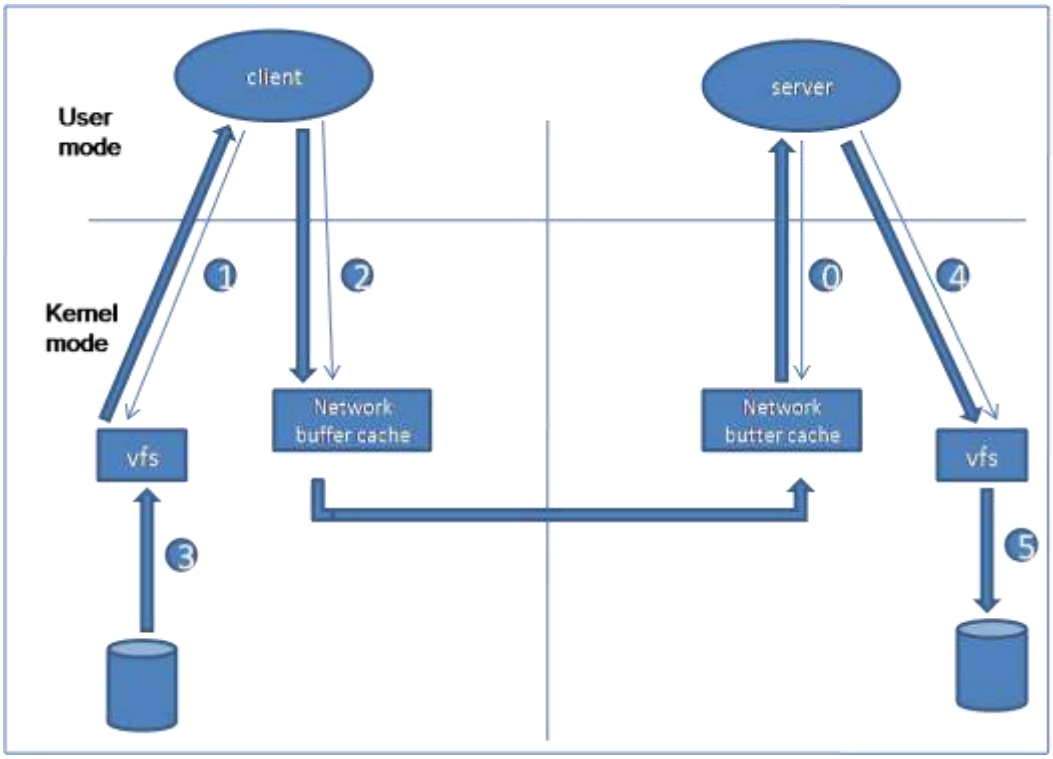

Fig. 5.2 File transfer between client and server with dgridOS services.

5.2 FREEDOM FROM RESTRICTED USAGE OF RESOURCES.

The middleware does not rule on how many resources a job can consume because there was no way for the administrator to specify how much or to what degree he wants to share his resources on a Grid. Now it is taken care by the dgridOS. The middleware have their own set of services defined, which could be the only means for programmers to code grid-application, thus bounds the freedom and capabilities of a programmer. Integrated inside in the operating system it will provide a better standard platform when compared to different grid middleware.

5.3 NEW RACE IN RESOURCE SHARING COMPUTING

This move will speed up the resource sharing computing. Better comfort and ease Since resource sharing and its use will be dominant in coming years of software engineering and computing infrastructure so it is very necessary that the modules which basically makes resources sharing and use working should be integrated inside the kernel

of operating system. In accordance with the most frequently used routines inside the OS kernel so that they can always be inside the main memory. These functionalities at lowest level will optimize every application running just over or any level over it. Thus dgridOS guides to modern era of computing benefits.

6. RELATED WORK

6.1 GRIDOS

GridOS[18] provides operating system services that support grid computing. It makes writing middleware easier and provides services that make a normal commodity operating system like Linux more suitable for grid computing. The services are designed as a set of kernel modules that can be inserted and removed with ease. The modules provide mechanisms for high performance I/O (gridos io), communication (gridos comm), resource management (gridos rm), and process management (gridos pm). These modules are designed to be policy neutral, easy to use, consistent and clean.

The following principles drive the design of GridOS. These principles derive from the fact that the toolkits like Globus require a common set of services from the underlying operating system.

Modularity. The key principle in GridOS is to provide modularity. The Linux kernel module architecture is exploited to provide a clean modular functionality.

Policy Neutrality. GridOS follows one of the guiding principles of design of operating systems: policy free mechanisms. Instead of providing a .black box. Implementation that takes care of all possibilities, the modules provide a policy-free API which can be used to develop high level services like GridFTP.

Universality of Infrastructure. GridOS provides a basic set of services that are common to prevalent grid software infrastructures. It should be easy to build radically different toolkits like Globus (a set of independent services) and Legion (an object-based meta-systems infrastructure).

Minimal Core Operating System Changes. We do not want to make extensive modifications to the core operating system as that would make it difficult to keep up with new OS versions.

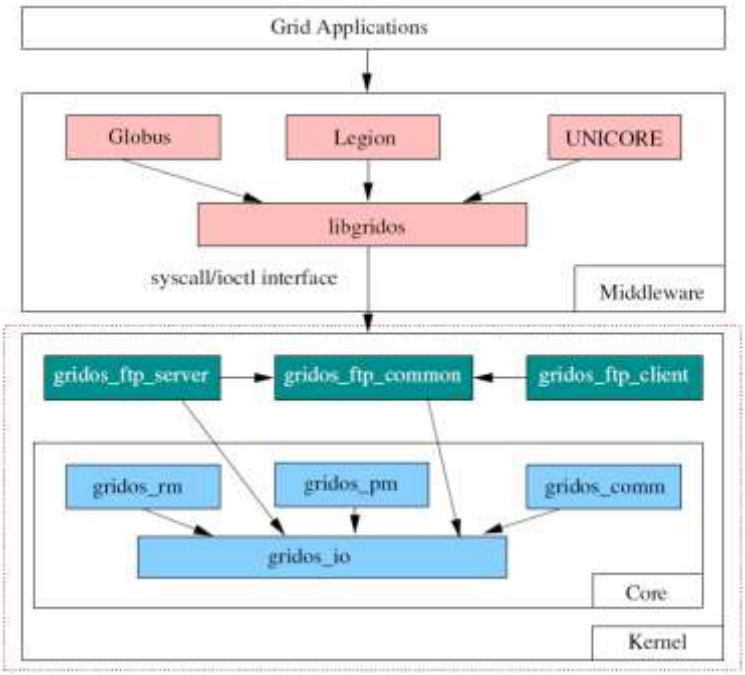

Fig. 6.1 Major modules in gridos

Additional Modules

1) *gridos ftp common:* This module provides facilities common to any FTP client and server. This includes parsing of FTP commands, handling of FTP responses etc.

2) gridos ftp server: This module implements a simple FTP server in kernel space. Due to its design, copying of buffers between user and kernel space is avoided. The server does not start a new process for each client as is usually done in typical FTP servers. This incurs low overhead and highperformance. The server also uses gridos io facilities to monitor bandwidth and adjust the _le system buffer sizes. The _le system buffers are changed depending on the _le size to get maximum overlap between network and disk I/O. The module is designed with security in mind. Even while operating in the kernel mode it drops all privileges and switches to an unprivileged user-id and group-id. It also heroots to FTP top level directory docroot which can be con_gured dynamically.

3) gridos ftp client: This module implements a simple FTP client in kernel. The main purpose of this module is to decrease the overhead of writing or reading _le on the client side. Our experiments indicate that primary overhead on the client side is the time spent

in reading and writing _les. By carefully optimizing the _le system buffers to achieve maximum overlap between network and disk I/O, high-performance is achieved.

6.2 VIGNE GRID OPERATING SYSTEM

They have presented the design of three self-healing services of the Vigne[15] Grid Operating System. The self-healing property is important to ensure the availability of the system and to relieve users and programmers from dealing with reconfigurations and failures. One of the main contributions of this architecture is the application management service which decentralizes application control and provides applications with generic and transparent fault-tolerance policies. Three self-healing services of Vigne, namely system membership, application management, and volatile data management.

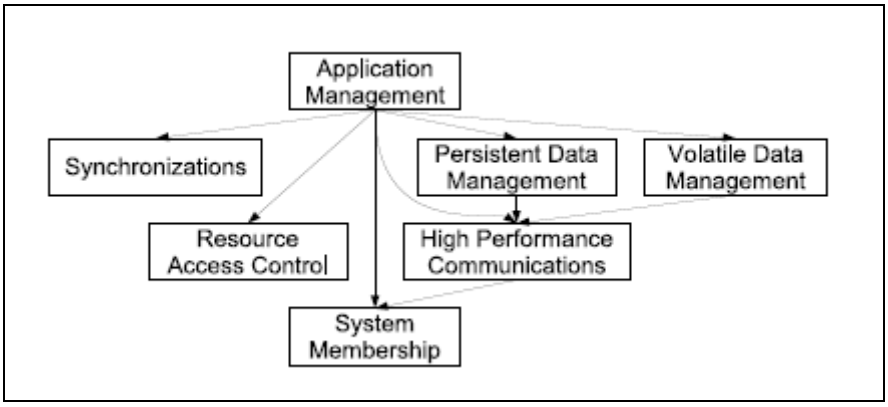

Fig. 6.2 Services of a Vigne Grid Operating System

6.3 XtreemOS

The overall objective of the XtreemOS[5] project is the design, implementation, evaluation and distribution of an open source Grid operating system (named XtreemOS) with native support for virtual organizations (VO) and capable of running on a wide range of underlying platforms, from clusters to mobiles. It propose an approach to investigate the design of a Grid OS, XtreemOS, based on the Linux existing general purpose OS. The architecture described in by XtreemOS is follows:

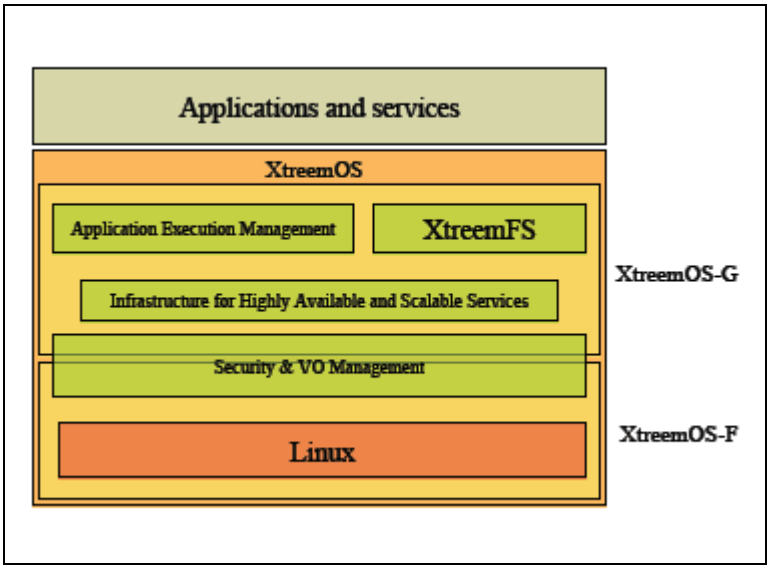

Fig. 6.3 XtreemOS architecture

7. CONCLUSION AND FUTURE WORK

The modular architecture presented covers all the core functionalities of grid. The modules and its routines can be visualized to a perfect virtual organization. A user can take advantage of grid resources of a virtual organization they belong to while having the illusion to execute their applications on their local computers.

Apart from the core benefits of grid computing, the advantages of grid enabled operating system is a matter of high importance, because it will shape the applications to new and better level.

The design of the dgridOS grid enabled operating system is a viable computing model for the coming years in distributed computing. It will explore the computing resources usage to a much optimized level.

Future Work

The approach presented here, shows a lot of potential for extensibility and improvement.

- One of the possible areas of research in the future is the security module explorations. However this module will be explored in future.
- Since it is implemented at the cost of memory requirement, research on memory - performance graphs in relation with different operating systems needs a high level concern.

8. REFERENCES

RESEARCH PAPERS and BOOKS

1. A. S. Grimshaw, W. A. Wulf, and the Legion team, "The legion vision of a worldwide virtual computer". *Communications of the ACM*, vol. 40, no. 1, pp. 39.45, Jan. 1997.
2. Abramson, D., Buyya, R., and Giddy, J."Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid",2004, from http://www.csse.monash.edu.au/~davida/papers/hpcasia.pdf
3. Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya, "Visual Modeler for Grid Modeling and Simulation (Grid Sim) Toolkit", 2003.
4. Bart Jacob, Luis Ferreira, Norbert Bieberstein, Candice Gilzean, Jean-Yves Girard, Roman strachowski, Seong Yu, "Enabling Application For Grid Computing Using Globus" ibm redbook, 2003.
5. Christine Morin, PARIS Project-team, "XtreemOS: a Grid Operating System Making Your Computer Ready for Partincipating in Virtual Organizations", 2007.
6. Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134-1706 USA, "Cluster Computing", 2004.
7. Dan Kusnetzky, Carl W. Olofson ,white paper on "Oracle 10g: putting Grids to Work" by, april 2004.
8. Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel, 2nd Edition", publisher O'Reilly,2002.
9. Foster, I., Frey, J., Livny, M., Tannenbaum, T. and Tuecke, S. (2001). "Condor-G: A Computation Management Agent for Multi-Institutional Grids". Proc. of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August, 2001.
10. Fran Berman, Geoffrey Fox and Tony Hey, "The Grid past present and future", 2005.
11. I. Foster, Argonne & U. Chicago, "Open Grid Services Architecture", jan-2005.
12. Ian Johnson, "Security Requirements for a Grid-based OS", jan-2007.

13. Irwin Boutboul, Dikran Meliksetian, Joe Zhou and others, "Grid Computing-Solutions Brief", ibm redbook, 2005.
14. Klaus krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems", 2003.
15. Louis Rilling. Vigne: "Towards a Self-Healing Grid Operating System". Dresden, Germany,August 2006.
16. Luis Ferreira, fabiano Lucchese, Tomoari Yasuda, Chin Yau Lee, Carlos Alexandre Queiroz, Elton Minetto, Antonio Mungioli, "Grid Computing in research and education", ibm redbook, 2005.
17. Manish Parashar, Craig A. Lee, "Special Issue on Grid Computing", 2005.
18. Pradeep Padala and Joseph N Wilson, "GridOS: Operating System services for grid architectures, 2003.
19. Rajkumar Buyya and Manzur Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling of Grid computing", 2002.
20. Silicon Graphics, sgi while paper on "The Future of Cluster Computing", 2005.
21. Srinivas Nimmagadda, Ilan Harari, "Scalability Issues in Cluster Computing Operating Systems".
22. Viktors Berstis, "Fundamentals of Grid Computing", ibm redbook, 2002.
23. White paper on "Oracle Database 10g Release 2: A Revovlution in Database Technology", may 2005.

WEBSITES:

i. http://www-03.ibm.com/grid/index.shtml
ii. http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14231/
iii. http://www.globus.org/
iv. http://www.teragird.org/
v. http://www.unicore.org/
vi. http://www.gridforum.org/
vii. http://www.gridtoday.com/

viii. http://www.sourceforge.net/
ix. http://www.buyya.com/cluster/
x. www.oasis-open.org
xi. www.w3c.org
xii. http://legion.virginia.edu
xiii. www.nsf-middleware.org
xiv. http://www.knoppix.org/
xv. http://www.csse.monash.edu.au/~davida/nimrod/index.htm
xvi. http://www.google.co.in

APPENDIX A

A1 GRID ORGANIZATIONS

World Community Grid

The World Community Grid's mission is to create the largest public computing grid benefiting humanity. The work is built on the belief that technological innovation combined with visionary scientific research and large-scale voluntaryism can change our world for the better. For more information, please refer to the following Web site:
http://www.worldcommunitygrid.org/

Globus Alliance

The Globus Alliance conducts research and development to create fundamental technologies for grid computing. The alliance is formed by a group of sponsors and collaborators from around the world. The core team is based at the Argonne National Laboratory and other worldwide institutions (see http://www.globus.org/about/team.html for more information).
The Globus Toolkit is being developed by the Globus Alliance and many others all over the world. For more information, please refer to the following Web site:
http://www.globus.org/

Global Grid Forum

The Global Grid Forum (GGF) is a community-initiated organization of thousands of active participants from industry and research. GGF's primary objective in this organization is to promote development, deployment, and implementation of grid technologies through creation of technical specifications, user experiences, and implementation guidelines. For more information, please refer to the following Web site:
http://www.gridforum.org/

OASIS and WSRF TC

Organization for the Advancement of Structured Information Standards (OASIS) is an international consortium that drives the development and adoption of e-business standards

and technologies. For more information, please refer to the following Web site: http://www.oasis-open.org/ The OASIS Web Services Resource Framework (WSRF) Technical Committee (TC) works on the definition of a generic, royalty-free, open framework for modeling and accessing stateful resources (required for grid computing) using Web services. For more information, please refer to the following Web site:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

W3C

The World Wide Web Consortium (W3C) is an international consortium that develops specifications and guidelines for Web technologies. For more information, please refer to the following Web site:
http://www.w3.org/

TeraGrid

TeraGrid is a initiative to build and deploy the world's largest, distributed infrastructure for open scientific research. It combinations three programs: National Science Foundation (NSF) Terascale initiative: Terascale Computing System (TCS®), Distributed Terascale Facility (DTF) and Extensible Terascale Facility (ETF). For more information, please refer to the following Web site:
http://www.teragrid.org/

A2 GENERATIONS OF DISTRIBUTED COMPUTING

General	Characteristics
First (Host-based Computing)	• Dumb Terminal. • Single Server. • Monolithic Applications.
Second (Remote Access)	• Single Client supporting only terminal emulation functions. • Single Server.

Third (Client/Server)	• Single Client supporting rules processing as well as user interface. • Up to two servers.
Fourth (Multitier)	• Single Client supporting rules processing as well as user interface. • More than two server tiers.
Fifth (Grid Computing)	• Virtual Environment where all systems are considered a pool of resources. • N-tier. • Service-Oriented Architectures.

Table A2 generations of distributed computing

Entering the Fifth Generation . Grid Computing

Grid Computing is the result of several trends coming together. Some of these are the following:

- New standards for object-to-object communications making it easier to build multivendor, multi-application networks.
- High-performance microprocessors have become available, making it possible to deploy large applications on a number of low-cost systems rather than a single mid-range system.
- High-speed networking technology is becoming both less costly and readily available, offering higher levels of performance when deploying distributed application architectures.

A3 IDC'S MODEL OF GRID COMPUTING

IDC's Model of Grid Computing

Type of Grid	Goals	Challenges	Description
Computational	• Raw application performance • Scalability • Reliability	• Not all applications can be segmented in this fashion • Emerging standards (OGSA)	• Multisystem configuration focused on high computational performance
Application	• Scalability • Robustness • Reliability • Availability	• Established applications may need to be rewritten • Not all applications can be segmented in this fashion	• Multisystem configuration focused on application scalability, reliability, and availability
Storage	• Scalability of storage • Reliability/availability of storage • Optimal use of storage devices • Increase interoperability	• Single vendor approaches • Single operating system approaches • Single data management approach	• Multisystem configuration focused on storage scalability, reliability, and availability
Optimization	• Scalability • Reliability • Optimal use of organization's resources allowing all systems to appear to be a single pool of available resources	• Sophisticated server provisioning and management software needed • Multisystem workload management required • Singe vendor approaches • Single operating system approaches • Single hardware architecture approaches	• Multisystem configuration focused on dynamic resource allocation, scalability, reliability, and manageability • A complete implementation of the Grid Computing concept

Fig. A3 IDC's model of grid computing

A4 PSEUDO CODE FOR DGRIDOS ENVIRONMENT

1. dgridOScore starts
    a. call dgrid_Initializer module
        i. [dgridOS_environment variables initializations].
        ii. create dgridLocalCacheBuffer(BufferStructure *ptr, int BufferInitialSize)
        iii. getComputationalResourceRating(int *compRat[], String *compType[], int compNumber)
        iv. getMemoryResourceSize(int memSize[], int memType[],int memNumber)
        v. getCommunicationChannelBandwith(int SourceNode_id, int DestNode_id, int bandwidth[])
        vi. checkSecurityStatus() set SECURITY_STATUS=ON;

vii. addRecordTOdgridLocalCacheBuffer([dgridOS_environment variables initializations], resName, resType and parameterValues, securityStatus)

b. call dgrid_buffMgr module

i. checkHardwareEventGenerator(Event_node_id, Event_loginUser)
if (both data matches) then call checkHardwareUpdate
else file_a_complain to head_node of fake event.

ii. checkHardwareUpdate()
if(there is any) then Update respective record in dgridLocalCacheBuffer()
else do nothing.

c. call dgrid_launcher

Thread 1:

i. call dgridOScom module
ii. call dgridOSsec module
iii. call dgridOSres module
iv. call dgridOSvfs module
v. call dgridOSpro module

Thread 2:

```
wait_for_a_dgridOSmoduleEvent()
switch(case){
case1: dgridOScom module event
        call com_module_service();
        wait_for_a_dgridOSmoduleEvent();
case2: dgridOSsec module event
        call sec_module_service();
        wait_for_a_dgridOSmoduleEvent();
case3: dgridOSres module event
        call res_module_service();
        wait_for_a_dgridOSmoduleEvent();
case4: dgridOSvfs module event
```

```
                    call vfs_module_service();
                    wait_for_a_dgridOSmoduleEvent();
                case5: dgridOSpro module event
                    call pro_module_service();
                    wait_for_a_dgridOSmoduleEvent();
                case6: dgridOS other event
                    call otherEvent_module_service();
                    wait_for_a_dgridOSmoduleEvent();
                case default:
                    wait_for_a_dgridOSmoduleEvent();
                }
```

Thread 3:

d. call dgridOScore_StartUpchecker()

//

```
//      dgridOSpro module //
```

2. dgridOSpro()
    a. strats jobListener()
        {
        waitJobOccurrence(Job_id, subJob_id, jobContextStructue)
        if (JobOccurred) then {
                collect JobContextStructure inside the dgrid_jobCache
                CheckWhetherAllResources are available on Local node;
                    if(yes)
                    {
                    sendAllocateRequest to dgrid_resMgr;
                    do the operation and collect the result from dgrid_jobCacheOutput section.
                    }
                    else
                    {
                    send ResourceRequest to head_node resMgr;

```
                    get the allocation
                    handover the control to processCtrl which maintains
                    Cache of migrated processes;
                    processCtrl return the result Cache to host node_id
                    }
            waitJobOccurrence(Job_id, subJob_id, jobContextStructue)
            }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```