Volume 2, No. 1, January 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

A SIMPLEX GRID COMPUTING METHODOLOGY FOR MONITORING RESOURCES

G. Mohammed Nazer*[1] and Dr.S.S.Jayachandran[2],

[1]Head, Dept of Computer Applications,
IFET College of Engineering, Villupuram, India.
kgmohammednazer@gmail.com

[2]Principal,
IFET College of Engineering, Villupuram, India.
ssjifet@gmail.com

Abstract – Controlling large scale distributed system is very difficult for the reliable operations of a system. This is because of two reasons. One, wide fluctuations in the availability of idle processor cycles. Second, communication latencies over multiple administrative domains. So, the quality of service in executing grid applications becomes a challenge. With such fluctuations there is no certainty when a grid task will complete its execution time and therefore become unpredictable. To overcome this problem, grid computing technology is used. More specifically a data grid is used to aggregate the unused storage space into a much large virtual data store. This is further configured to achieve improved performance and reliability. Thus it has the possibility of substantially increasing the efficiency of resource usage. A three-tier architecture of Distributed Computing Grid Model (DCGM) is brought forward. It acts as the infrastructure of multi-site scheduling environment in which Distributed scheduling server is utilized. First Come First Serve (FCFS) algorithm is implemented. Thus the applications of splitting the numbers depending upon the CPU idle rate is established and thereby reducing the execution time.

Index : Distributed system; Grid; Applications; Grid Computing; Virtual data store; Autonomic Computing.

INTRODUCTION

A Grid is all about gathering together resources and making them accessible to users and applications. It is "a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to computational capabilities". The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing.

Exploiting underutilized resources: The easiest use of grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the grid.

In most organizations, there are large amount of underutilized computing resources. Most desktop machines are busy less than 5% of the time. In some organizations, even the server machines can often be relatively idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage.

The processing resources are not the only ones that may be underutilized. Often, machines may have enormous unused disk drive capacity. Grid computing, more specifically, a "data grid", can be used to aggregate this unused storage into a much larger virtual data store, possibly configured to achieve improved performance and reliability over that of any single machine.

Resource balancing and Reliability: Another function of the grid is to better balance resource utilization. A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization. High-end conventional computing systems use expensive hardware to increase reliability.

A. Organization of the Paper

In Section II we present the formal Grid Computing model that we use to overcome the unpredictable execution time. In particular, we use a data grid to aggregate the unused storage space into a much large virtual data store. In Section III we present how to schedule the grid, user's perspective, Grid configuring and implementation. In Section IV we discuss about Monitoring resources, its progress, recovery and testing. Finally, in Section V we show various experimentation results and in Section VI we conclude with results and discuss directions for future

© JGRCS 2010, All Rights Reserved

work.

A FORMAL GRID COMPUTING MODEL

Grid Computing Model: Although coordinated use of resources is not a trivial problem in a closed environment, it gets more complicated when it is attempted across geographical and organizational boundaries. To overcome the system problems such as identity and authentication, authorization and policy, resource discovery, resource allocation, resource management and security, a set of protocols and mechanisms need to be defined that address the security and policy concerns of the resource owners and users. In addition to grid protocols that have to be defined, a set of grid applications programming interfaces (API's) and software development toolkits (SDK's) need to be defined. They provide interfaces to the grid protocols and services as well as facilities applications development by supplying higher-level abstraction.

Grid architecture is to identify the requirements for general classes of component. The result is an extensible, open architectural structure within which can be placed solutions to key virtual organization requirements. Components within each layer share common characteristic but can build on capabilities and behaviors provided by any lower layer.

Access to additional resources: 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. The additional resources can be provided in additional numbers and/or capacity. The grid can enable more elaborate access, potentially to remote medical diagnostic and robotic surgery tools with two-way interaction from a distance. The variations are limited only by one's imagination.

Resource balancing: A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. This feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization. This can happen in two ways:

- 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.

Load balancing: 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. This can potentially reduce communication and other forms of contention in the grid.

Reliability: High-end conventional computing systems use expensive hardware to increase reliability. They are built using chips with redundant circuits that vote on results and contain much logic to achieve graceful recovery from an assortment of hardware failures. The machines also use duplicate processors with hot pluggability so that when they fail, one can be replaced without turning the other off. Power supplies and cooling systems are duplicated. The systems are operated on special power resources that can start generators if utility power is interrupted. All of this builds a reliable system, but at a great cost, due to the duplication of high-reliability components.

An alternate approach to the reliability is to relay more on *software technology* than expensive hardware. A grid is just the beginning of such technology. The systems in a grid can be relatively inexpensive and geographically dispersed. Thus, 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. Grid management software can automatically resubmit the jobs to other machines on the grid when a failure is detected and their results can be checked for any kind of inconsistency, such as computer failures, data corruption or tampering.

Such grid systems will utilize *"autonomic computing"* which is a type of software that automatically heals problems in the grid, perhaps even before an operator or manager is aware of them.

Management: The goal to virtualize the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manager a larger, more disperse IT infrastructure. It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a large organization. The grid offers management of priorities among different projects.

Connectivity – Communicating easily and securely: This defines core communication and authentication protocols required for grid specific network transactions. These utilize the existing Internet protocols such as IP, Domain Name Service, various routing protocols such as BGP and so on. Another protocol namely Grid Security Infrastructure provides uniform authentication, authorization and message protection mechanism. Communication protocols enable the exchange of data between fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources.

Resource – Sharing single resources: The resource layer defines protocols required to initiate and control sharing of local resources. It builds on connectivity layer communication and authentication protocols for the secure negotiations, initiates monitoring, control, accounting and payment of sharing operations on individual resources. Protocols defined at this layer include:

- Grid Resource Allocation Management (GRAM) – Remote Allocation, reservation, monitoring and control of resources.
- Grid File Transfer Protocol (GFTP) – High performance data access and transport.
- Grid Resource Information Service (GRIS) – Access to structure and state information

© JGRCS 2010, All Rights Reserved

Collective – Coordinating multiple resources: This collective layer contains protocols and services that are not associated with any one specific resources but rather are global in nature and capture interactions across collections of resources. Because collective components build on the narrow resources can implement a wide variety of sharing behaviors without placing new requirements on the resources being shared.

Applications: The final layer in the grid architecture comprises the applications that operate within a virtual organization environment. Applications are constructed in terms of and by calling upon services defined at any layer. At each layer we have well defined protocols that provide access to some useful service resource management, data access and resource discovery.

A. Characteristics of resources used in the Grid

The resources in a grid typically share at least some of the following characteristics:

- They are numerous.
- They are owned by different organizations and individuals.
- They have different security requirements and policies.
- They are heterogeneous.
- Connected by multilevel heterogeneous networks.
- They have different resource management policies.
- They are likely to be separated geographically.

Several packages sensed monitoring data and made it available to a distributed framework of services and client tools. The set of information providers deployed was determined by identifying and prioritizing desirable grid-level (such as overall resource availability and consumption) and Virtual Organization (VO) level. Other requirements derived from auditing, scheduling and debugging considerations. The framework was built by integrating existing monitoring software tools into a simple way.

B. Resources Selection and Management

Resource Types: A grid is a collection of machines, sometimes referred to as "nodes", "resources", "members", "donors", "clients", "hosts", "engines" and many other terms. They all contribute any combination of resources to the grid as a whole. Some resources may be used by all users of the grid while others may have specific restrictions.

Computation: Computing cycles is one of the most common resource provided by the processors of the machines on the grid. The processors may vary in its speed, architecture, software platform and other associated factors, such as memory, storage and connectivity.

Storage: A grid providing an integrated view of data storage is sometimes called as a "data grid". Each machine on the grid usually provides some quantity of storage for grid. Storage can be of 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 to serve as temporary storage for running applications.

Communications: A grid uses software to help jobs to communicate with each other. An application may be split itself into a large number of sub-jobs. Each of these sub-jobs is a separate job in a grid. However, the application may implement an algorithm, that requires that the sub-jobs communicate some information among them. The sub-jobs need to be able to locate other specific jobs, establish a communication connection with them, and send the appropriate data.

Observation, management and measurement: Usually the donor software will include some tools that measure the current load and activity on a given machine using either operating system facilities or by direct measurement using load sensor. Some gird systems provide the means for implementing custom load sensors for other than CPU or storage resources. Such measurement information is useful not only for scheduling, but also for discovering overall usage patterns in the grid. The statistics can show trends which may signal the need for additional hardware. Also, measurement information about specific job can be collected ad used to better predict the resource requirements of that job the nest time it is run.

The better the prediction, the more efficiently the grids workload can be managed. The measurement information can also be saved for accounting purposes, to form the basis for grid resource brokering or to manage priorities more fairly.

Software and licenses: The grid 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. When the licensing fees are significant, this approach can save significant expenses for an organization.

Special equipment, capacities, architectures and policies: Platforms on the grid will often have different architecture, operating systems, devices, capacities and equipment. Each of these items represents a different kind of resource that he grid can use as criteria for assigning jobs to machines. While some software may be available on several architectures, they are 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.

Jobs and Applications: Although various kinds of resources on the grid may be shared and used, they are usually accessed via an executing "application" or "job". Usually we use the term "application" as the highest level of a piece of work the grid. Jobs are programs that are executed at an appropriate point on the grid. They may compute something, execute one or more system commands, move or collect data or operate a machinery. 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.

© JGRCS 2010, All Rights Reserved

Selection of Resources: The various steps included in the selection of various resources in a grid are:

- Check the system status
- Read the CPU Idle Rate
- System status if enable, using the commands (Iostat and Vmstat) we retrieve CPU Idle Rate.
- Check the available memory.

Large software systems are difficult to build in a timely manner by just collecting requirements, analyzing the problem, and then partitioning the many functional components to programmers and finally integrating and testing modules built by multiple programmers. This waterfall-style development of new software systems from the ground up is increasingly untenable as the scale of software grows. In the near future large systems will typically be composed using libraries and existing legacy code to reduce the development risk and cost.

Resource Management: Grid systems allow applications to assemble and use collections of resources on an as-needed bass, without regard to its physical location. Grid middleware and other software architecture that manage resources have to locate and allocate resources according to application requirements. They also have to manage other activities like authentication and process creation that are required to prepare a resource to use. Figure 2.1 shows the various resource selection process.

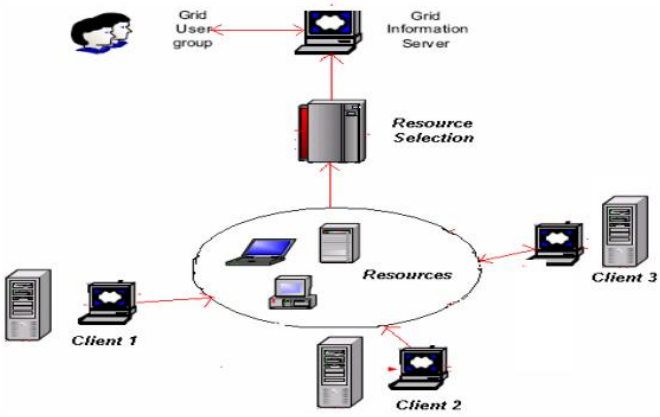

Figure 2.1 Resource Selection

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.

Various tools may be able to identify important trends throughout the grid, informing management of those that require attention.

C. Building a Grid

How to build the grid? 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. Figure 2.2 shows such a simple grid architecture.

A grid may grow to cross organization boundaries and may be used to collaborate on projects of common interest, which is referred to as "Intergrid". The highest levels of security are usually required in this configuration to prevent possible attacks and spying. The "Intragrid" offers the prospect for trading or brokering resources over a much wider applications.

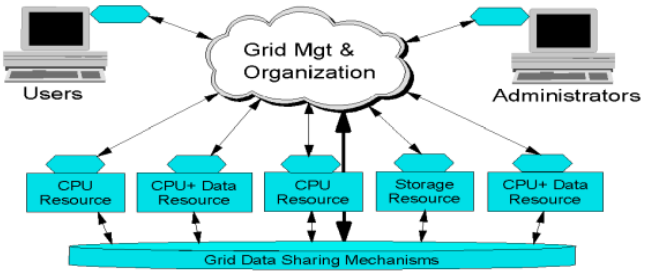

Figure 2.2 A Simple Grid Architecture

Deployment Planning: The use of a grid is often born from a need for increased resources of some type. One often looks to their neighbor who may have excess capacity in the particular resource. One of the first considerations is the hardware available and how it is connected via a LAN or WAN. Next, an organization may want to add additional hardware to augment the capabilities of the grid. It is important to understand the applications to be used on the grid. Their characteristics can affect the decisions of how to best choose and configure the hardware and its data connectivity.

Security: Security is a much more important factor in planning and maintaining a grid than in conventional distributed computing, where data sharing comprises the bulk of the activity. In a grid, the member machines are configured to execute programs rather than just move data. This makes an unsecured grid potentially fertile ground for viruses and Trojan horse programs. For this reason, it is important to understand exactly which components of the grid must be rigorously secured to deter any kind of attack. Furthermore, it is important to understand the issues involved in authenticating users and properly executing the responsibilities of a certificate authority.

Organization: The technology considerations are important in deploying a grid. However, organizational and business issues can be equally important. It is important to understand how the departments in an organization interact, operate, and contribute to the whole. Often, there are barriers built between departments and projects to protect their resources in an effort to increase the probability of timely success.

However, by rethinking some of these relationships, we can find that more sharing of resources can sometimes benefit the entire organization better. For example, a project that finds itself behind schedule and over budget may not be able to afford the resources required to solve

© JGRCS 2010, All Rights Reserved

the problem. A grid would give such projects an added measure of safety, providing an extra margin of resource capacity needed to finish the project. Similarly, a project in its early stages, when computing resources are not being fully utilized, may be able to donate them to other projects in need.

Management Components: Any grid system has some management components. First, there is a component that keeps track of the resources available to the grid and which users are members of the grid. This information is used primarily to decide where grid jobs should be assigned. Second, there are measurement components that determine both the capacities of the nodes on the grid and their current utilization rate at any given time. This information is used to schedule jobs in the grid. Such information is also used to determine the health of the grid, alerting personnel to problems such as outages, congestion, or over commitment. This information is also used to determine overall usage patterns and statistics, as well as to log and account for usage of grid resources. Third, advanced grid management software can automatically manage many aspects of the grid. This is known as "autonomic computing," or "recovery oriented computing." Software would automatically recover from various kinds of grid failures and outages, finding alternative ways to get the workload processed.

Donor Software: Each machine contributing resources typically needs to enroll as a member of the grid and install some software that manages the grid's use of its resources. Usually, some sort of identification and authentication procedure must be performed before a machine can join the grid. A certificate authority can be used to establish the identity of the donor machine as well as the users and the grid itself. Some grid systems provide their own login to the grid while others depend on the native operating systems for user authentication.

In the latter case, a user ID mapping system may be needed to match the user's rights properly on different machines, which is manually maintained by a grid administrator. He determines which user ID a given user may possess on each grid machine and enters these IDs in a protected database or registry. In this way, when grid jobs are submitted to different machines for a user, the proper local machine user ID is used for determining the users rights. In some grid systems, it is possible to join the grid without any special authentication.

Software Submission: Usually any member machine of a grid can be used to submit jobs to the grid and initiate grid queries. However, in some grid systems, this function is implemented as a separate component installed on "submission nodes" or "submission clients." When a grid is built using dedicated resources rather than scavenged resources, separate submission software is usually installed on the user's desktop or workstation.

Distributed Grid Management: Larger grids may have hierarchical or other type of organizational topology usually matching the connectivity topology. That is, machines locally connected together with a LAN form a "cluster" of machines. The grid may be organized in a hierarchy consisting of clusters of clusters.

The work involved in managing the grid is distributed to increase the scalability of the grid. The collection and grid operation and resource data as well as job scheduling is distributed to match the topology of the grid. For example, a central job scheduler will not schedule a submitted job directly to the machine which is to execute it. Instead the job is sent to a lower level scheduler, which handles a set of machines (or further clusters). The lower level scheduler handles the assignment to the specific machine. Similarly, the collection of statistical information is distributed. Lower level clusters receive activity information from the individual machines, aggregate it, and send it to high level management nodes in the hierarchy.

GRID SCHEDULING

Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. A grid scheduler must make a resource selection in an environment where it has no control over the local resources, the resources are distributed and information about the system is often limited.

A Grid is a distributed collection of computer and storage resources maintained to serve the needs of some community or virtual organization (VO). Any of the potentially large number of authorized users within that VO has access to all or some of these resources and is able to submit jobs to the Grid and expect responses. The choice of algorithms used to schedule jobs in such environments depends on the target application.

Scheduling strategically allocates the CPU to a process based on specified criteria. There are many different methods of selecting which process will be given control of the CPU. Each of these methods follows a different scheduling algorithm and has advantages and disadvantages.

Purpose of scheduling: The purpose of scheduling is to maximize the utilization of the CPU. In order to do this, a process should be running at all times. For example, if a process is waiting for some event to occur before it is able to continue execution, then it should not have control of the CPU. If it does, then the CPU is being wasted.

Types of scheduling: There are three main scheduling approaches for achieving predictable execution times in grids.

Advanced Reservations: Used in silver and interleaved hybrid scheduling, aims to build mechanisms for requesting exclusive use of a portion of the capacity of a resource for grid jobs, and therefore avoid having fluctuating shared resource capacities. The drawback of this approach is that not all local schedulers provide advanced reservation support.

© JGRCS 2010, All Rights Reserved

Predictive Techniques: Predictive techniques are implemented in the network weather service. It uses statistical extrapolation of historical measurement data to forecast processor utilization and network bandwidth.

Grid schedulers use this information to estimate application execution time and perform scheduling. Apples and every ware are examples of predictive technique-based systems. The predictive approach assumes that performance data of past execution of applications on known resources are available. This assumption however is not practical since there are scenarios where resource joined and leave the grid on at any time, or where there are new applications submitted for execution. This technique is also difficult to implement since most current systems do not provides the additional information required by predictive techniques.

Feedback Control: Feedback control is an approach widely used in engineering to updating high performance in the presence of uncertainty. The key idea is to compare measured with the desired performance and to perform corrections dynamically. CHAIMS a service-centered grid represents local resources as service providers and grid tasks are represented as service request, is one system using feedback control it monitors service fulfillment progress and uses the information to repair schedulers. Grad soft is another system using feedback control approach. It providers an application development environment for building grid jobs that can automatically adapt to changing grid environments.

The feedback control approach has several advantages over advanced reservation and predicated techniques. First, it requires no additional support from local schedulers. Hence, a grid scheduler implementing this approach can interface with any local scheduler. Second, it does not require data of past application executions. Therefore, a grid scheduler can provide predictable execution times for the grid applications that have not been previously executed.

Despite the advantages of the feedback control scheduling approach, such systems require extensive use of application development and profiling tools. Both CHAIMS and Grad soft use complex methods for application performance monitoring and schedule connections. This results in complex frameworks for application development. These frameworks include special compilers and application toolkits that might be difficult for new application developers to learn and use.

Schedulers: Most grid systems include some sort of job scheduling software. This software locates a machine on which to run a grid job that has been submitted by a user. In the simplest cases, it may just blindly assign jobs in a round-robin fashion to the next machine matching the resource requirements. However, there are advantages to using a more advanced scheduler. Some schedulers implement a job priority system. This is sometimes done by using several job queues, each with a different priority. As grid machines become available to execute jobs, the jobs are taken from the highest priority queues first. Policies of various kinds are also implemented using schedulers.

Policies can include various kinds of constrains on jobs, users, and resources. For example, there may be a policy that restricts grid jobs from executing at certain times of the day.

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. More advanced schedulers will monitor the progress of scheduled jobs managing the overall work-flow. If the jobs are lost due to system or network outages, a good scheduler will automatically resubmit the job elsewhere. However, if a job appears to be in an infinite loop and reaches a maximum timeout, then such jobs should not be rescheduled. Typically, jobs have different kinds of completion codes, some of which are suitable for re-submission and some of which are not. Reserving resources on the grid in advance is accomplished with a "reservation system".

It is more than a scheduler. It is first a calendar based system for reserving resources for specific time periods and preventing any others from reserving the same resource at the same time. It also must be able to remove or suspend jobs that may be running on any machine or resource when the reservation period is reached.

- **Establishing a connection to a service** - The primitives setup and terminate all establish and end the connection from a client to a service.
- **Estimation** - Estimate allows a client to ask a service for cost estimates for a specific invocation. It is available both prior and during execution. The output is to be a name-tuple list (name of the cost factor, value of the cost factor and its uncertainty). If the service cannot provide estimates, its wrapper returns as reasonable values as possible, perhaps based on past executions. Cost estimates allow the composer or a CLAM optimizer to choose among alternative services and/or optimal execution paths.
- **Executing service methods** - Methods are executed by the following four calls. *Invoke*, *examine*, *extract* and *terminate*. Invoke starts the execution of a method, which then proceeds asynchronously; multiple, invokes with different parameters can occur within a single setup.

A. Using a Grid – A User's Perspective

In this section, we describe the typical usage activities in using the grid from an user's perspective.

Enrolling and Installing a Grid Software: A user first enrolls as a grid user, and installs the provided grid software on its own machine. He may optionally enroll his machine as a donor on the grid. Enrolling in the grid may require authentication for security purposes. The user positively establishes his identity with a certificate authority. This should not be done solely via the Internet.

© JGRCS 2010, All Rights Reserved

The certificate authority must take steps to assure that the user is in fact who he claims to be.

The certificate authority makes a special certificate available to software needing to check the true identity of a grid user and his grid requests. Similar steps may be required to identify the donating machine. The user has the responsibility of keeping his grid credentials secure. Once the user and/or machine are authenticated, the grid software is provided to the user for installing on his machine for the purposes of using the grid as well as donating to the grid. This software may be automatically pre-configured by the grid management system to know the communication address of the management nodes in the grid and user or machine identification information.

In this way, the installation may be a one click operation with a minimum of interaction required on the part of the user. In less automated grid installations, the user may be asked to identify the grid's management node and possibly other configuration information. He may choose to limit the resources donated to the grid, the times that his machine is usable by the grid, and other policy related constraints. The user may also need to inform the grid administrator which user IDs are his on other machines that exist on the grid.

Logging onto the Grid: To use the grid, most grid systems require the user to log on to a system using a user ID that is enrolled in the grid. Other grid systems may have their own grid login ID separate from the one on the operating system. A grid login is usually more convenient for grid users. It eliminates the ID matching problems among different machines. To the user, it makes the grid look more like one large virtual computer rather than a collection of individual machines. Globus, for example, implements a proxy login model that keeps the user logged in for a specified amount of time, even if he logs off and back on the operating system and even if the machine is rebooted. Once logged on, the user can query the grid and submit jobs. Some grid implementations permit some query functions if the user is not logged into the grid or even if the user is not enrolled in the grid.

Queries and Submitting Jobs: The user usually perform some queries to check to see how busy the grid is, to see how his submitted jobs are progressing, and to look for resources on the grid. Grid systems usually provide command line tools as well as graphical user interfaces (GUIs) for queries. Command line tools are especially useful when the user wants to write a script that automates a sequence of actions. For example, the user might write a script to look for an available resource, submit a job to it, watch the progress of the job, and present the results when the job has finished. Job submission usually consists of three parts, even if there is only one command required.

First, some input data and possibly the executable program or execution script file are sent to the machine to execute the job. Sending the input is called "staging the input data." Alternatively, the data and program files may be pre-installed on the grid machines or accessible via a mountable networked file system. When the grid consists of heterogeneous machines, there may be multiple executable program files, each compiled for the different machine platforms on the grid.

A nice feature provided by some grid systems is to register these multiple versions of the program so that the grid system can automatically choose a correctly matching version to the grid machine that will run the program. Some grid technologies require that the program and input data be first processed or "wrappered" in some way by the grid system.

This may be done to add protective execution controls around the application or just too simply collect all of the data files into one.

Second, the job is executed on the grid machine. The grid software running on the donating machine executes the program in a process on the user's behalf. It may use a common user ID on the machine or it may use the user's own user ID, depending on which grid technology is used. Some grid systems implement a protective "sandbox" around the program so that it cannot cause any disruption to the donating machine if it encounters a problem during execution. Rights to access files and other resources on the grid machine may be restricted.

Third, the results of the job are sent back to the submitter. In some implementations, intermediate results can be viewed by the user who submitted the job. In some grid technologies that do not automatically stage the output data back to the user, the results must be explicitly sent to the user, perhaps using a networked file system.

Scripts are also useful for submitting a series of jobs, for a parameter space application, for example. Some computation problems consist of a search for the desired result based on some input parameters. The goal is to find the input parameters that produce the best desired result. For each input parameter, a separate job is executed to find the result for that value. The whole application consists of many such jobs, which explore the results for a large number of input parameter values. Scripts are usually used to launch the many sub-jobs, each receiving their own particular parameter values. Parameter inputs can sometimes be more complex than simply a number. Sometimes a different input data set represents the "input parameter." Scripts help automate the large variety of more complex parameter space study problems.

For simpler parameter space inputs, some grid products provide a GUI to submit the series of sub-jobs, each with different input parameter values. When there are a large number of sub-jobs, the work required to collect the results and produce the final result is usually accomplished by a single program, usually running on the machine at the point of job submission. If there are a very large number sub-jobs required for an application, the work of collecting the results might be distributed as well.

B. Grid Configuring And Implementation

The data accessed by the grid jobs may simply be staged in and out by the grid system. However, depending on its size and the number of jobs, this can potentially add up to a large amount of data traffic. For this reason, some

© JGRCS 2010, All Rights Reserved

thought is usually given on how to arrange to have the minimum of such data movement on the grid.

For example, if there will be a very large number of sub-jobs running on most of the grid systems for an application that will be repeatedly run, the data they use may be copied to each machine and reside until the next time the application runs. This is preferable to using a networked file system to share this data, because in such a file system, the data would be effectively moved from a central location every time the application is run. Thus is true unless the file system implements a caching feature or replicates the data automatically.

There are many considerations in efficiently planning the distribution and sharing of data on a grid. This type of analysis is necessary for large jobs to better utilize the grid and not create unnecessary bottlenecks.

IV. MONITORING PROGRESS AND RECOVERY

The user can query the grid system to see how his application and its sub-jobs are progressing. When the number of sub-jobs becomes large, it becomes too difficult to list them all in a graphical window. Instead, there may simply be a one large bar graph showing some averaged progress metric. It becomes more difficult for the user to tell if any particular sub job is not running properly.

A grid system, in conjunction with its job scheduler, often provides some degree of recovery for sub-jobs that fail. A job may fail due to a:

- Programming error - The job stops part way with some program fault.
- Hardware or power failure - The machine or devices being used stop working in some way.
- Communications interruption - A communication path to the machine has failed or is overloaded with other data traffic.
- Excessive slowness

The job might be in an infinite loop or normal job progress may be limited by another process running at a higher priority or some other form of contention. It is not always possible to automatically determine if the reason for a job's failure is due to a problem with the design of the application or if it is due to failures of various kinds in the grid system infrastructure.

Schedulers are often designed to categorize job failures in some way and automatically resubmit jobs so that they are likely to succeed, running elsewhere on the grid. In some systems, the user is informed about any job failures and the user must decide whether to issue a command to attempt to rerun the failed jobs.

Grid applications can be designed to automate the monitoring and recovery of their own sub-jobs using functions provided by the grid system software application programming interfaces (APIs) as mentioned in Figure 4.1.

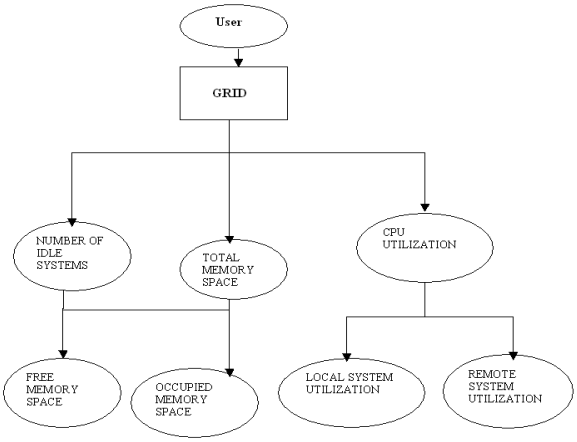

Figure 4.1 Automation of Monitoring and Recovery

Reserving Resources: To improve the quality of a service, the user may arrange to reserve a set of resources in advance for his exclusive or high priority use. A calendaring system analogy can be used here. Such a reservation system can also be used in conjunction with planned hardware or software maintenance events, when the affected resource might not be available for grid use.

In a *scavenging grid*, it may not be possible to reserve specific machines in advance. Instead, the grid management systems may allocate a larger fraction of its capacity for a given reservation to allow for the likelihood of some of the resources becoming unavailable. This must be done in conjunction with tools that have profiled the grid's workload capacity sufficiently to have reliable statistics about the grid's ability to serve the reservation.

Traditionally, the definition of task scheduling is the assignment of a set of tasks to some certain resources by means of starting and ending time of tasks, subject to certain constraints. However, computational grid involves so many resources over multiple administrative domains that appropriate resources should be selected carefully in order to provide the best Qos.

Thus, the traditional scheduling model based on static resources can not satisfy the large-scale dynamic resources requirement of the grid computing. In this paper, a new scheduling model oriented to the distributed computational grid is put forward.

Our *scheduling model* is new that it deals with multiple sites. Normally, resource selection algorithms can be classified into single-site and multi-site resource selection algorithms. Currently, most of the scheduler systems adopt the single-site resource selection algorithm such as Matchmaker/Class Ad system of University of Wisconsin-Madison, Nimrod/G Scheduler of Monash University, Silver Grid scheduler of Supercluster Organization and the Metascheduler of the Poznan Supercomputing and Networking Center.

In this paper, we propose our *enhanced multi-site resource selection algorithm* — FCFS algorithm, based on the distributed computational grid model and the grid scheduling model. The FCFS algorithm integrates a new density-based grid resource-clustering algorithm into the decoupled scheduling approach of the DCGM and decreases its time complexity. Also, we establish a performance model and mapping strategy for the

© JGRCS 2010, All Rights Reserved

synchronous iterative applications and demonstrate the correctness and effectiveness of FCFS algorithm in our simulation environment and the campus grid platform.

A. Models

Distributed Computational Grid Model (DCGM): - Distributed Computing Grid Model adopts three-level architecture as shown in Fig: the top level consists of Grid Information Servers (GIS) and Grid Meta Scheduling Server (GMSS); the second level has several domains and each domain consists of a Grid Distributed Scheduling Server (GDSS); all kinds of Grid Computing Resources (GCR) and Grid User Groups (GUG) are the third level.

- *Grid Information Servers* (GIS) are the essential part of any Grid software infrastructure, providing fundamental mechanisms for discovery and monitoring [10]. Each domain is controlled at least by one GIS, which dynamically collects information about the registered resources and spreads information to other GISs. A GIS receives Grid information request sent by GMSS, GDSS and GUG, finds proper resources and then returns the satisfactory resource aggregate to the requester.
- *Grid Meta Scheduling Servers* (GMSS) focus on harmonizing the scheduling of different GDSSs. The goal of the GDSS is to avoid distributing the same resources to two applications when they are submitted simultaneously. Every GMSS accepts the meta scheduling request from the GDSSs and cooperates with other GMSSs to increase the system throughput. We only focus on the GDSS scheduling policies, although more work needs to be done for GMSS policies.
- *Grid Distributed Scheduling Servers* (GDSS) are the key component in the architecture, which administers the efficient use of registered resources and maps the grand-challenging applications on the selected resources aggregate. The quantity of the GDSSs in one domain depends on its scale. When the Grid User Group (GUG) submits a job to GDSS, the GDSS contacts with the GIS to gather the information of the Grid. Then, the GDSS makes use of the decision module for scheduling and sends the meta scheduling request to the GMSS. Finally, it dispatches the tasks to the selected GCRs and retrieves the running result for the GUG.
- *Grid Computational Resources* (GCR) is no dedicated workstations or personal computers, which may be homogeneous or heterogeneous. Every GCR registered to GIS acts as the target of task mapping performed by GDSS or GMSS.
- *Grid User Group* (GUG) has challenging problems such as fluid dynamics, weather modeling and nuclear simulation etc. GUG interacts with Grid environment through the Grid Portal along with GDSS. On the whole, the purpose of our model focuses on constructing "The Poor Man's Supercomputer" as SETI@home and Data Synapse's Live Cluster did, which provides access to supercomputer level processing power with a fraction of the cost of a typical supercomputer.

In this paper, we use *resource selection algorithm*, under which we select the resources according to their CPU idle time using First – come First – serve (FCFS) algorithm and scheduling is done by Multilevel Feedback Queue Scheduling algorithm (MFQS), then we schedule the job according to the algorithm. We do the application as separating the job and dividing the numbers according to the resources.

B. Testing Methods

The various testing methods that are used in this paper are System testing, Validation Testing and Performance Testing.

System Testing: It checks system status and available memory.

Validation Testing: It checks CPU Idle rate, system status & available memory rate.

Performance Testing: It checks the performance of the system & CPU etc.

EXPERIMENT RESULTS

The following Figure 5.1 shows the various snap shots of different client information by using Multisite Resource Selection and Scheduling Algorithm using Grid computing.

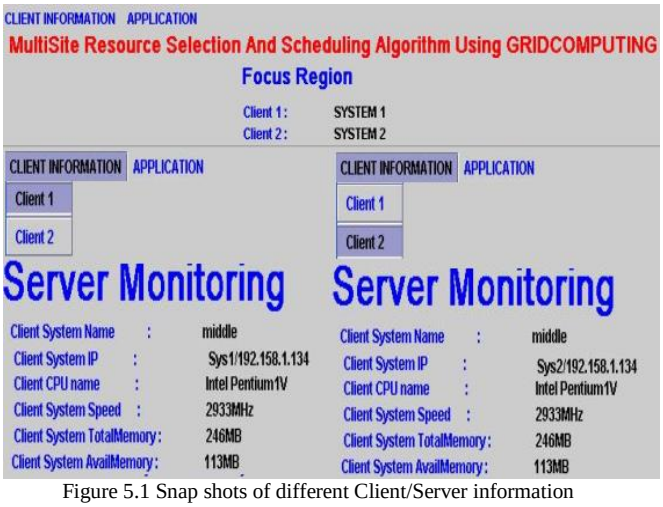

Figure 5.1 Snap shots of different Client/Server information

Figure 5.2 shows the snap shot of Encryption and Decryption Process completed.

© JGRCS 2010, All Rights Reserved

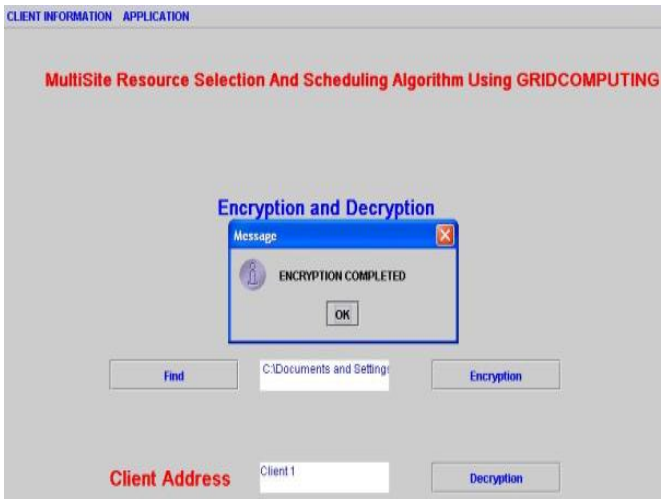

Figure 5.2 Snap shots showing Encryption Process.

Figure 5.2.1 Snap shots showing Decryption Process.

Table 5.1 and 5.2 shows the various test results of resource utilization of homogeneous systems in the Grid.

ID	System Name	System Status	Processor Name	CPU Speed	Total Memory	Available Memory
1	mid 43	Running	Intel Pentium	9233 MHZ	246 MB	86 MB
2	mid 42	Running	Intel Pentium	9343 MHZ	280 MB	52 MB
3	mid 44	Running	Intel Pentium	9244 MHZ	281 MB	52 MB
4	mid 45	Running	Intel Pentium	9811 MHZ	280 MB	50 MB
5	mid 40	Running	Intel Pentium	9233 MHZ	240 MB	92 MB
6	mid 41	Running	Intel Pentium	9322 MHZ	256 MB	76 MB

Table 5.1 Test results of resource utilization

ID	System Name	System Status	Processor Name	CPU Speed	Total Memory	Available Memory
1	mid 43	Running	Intel Pentium	9233 MHZ	246 MB	86 MB
2	mid 42	Running	Intel Pentium	9343 MHZ	280 MB	52 MB
3	mid 44	Running	Intel Pentium	9244 MHZ	281 MB	52 MB
4	mid 45	Running	Intel Pentium	9811 MHZ	280 MB	50 MB
5	mid 40	Stop				
6	mid 41	Running	Intel Pentium	9322 MHZ	256 MB	76 MB

Table 5.2 Test results of resource utilization

Figure 5.3 shows the graphical view of our test results.

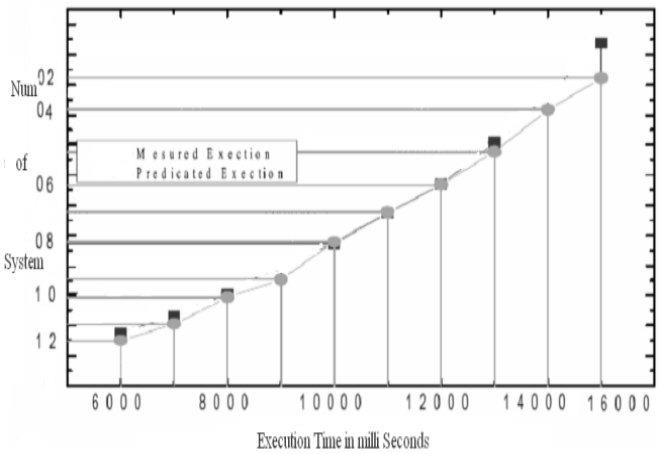

Figure 5.3 Graphical view of test results

CONCLUSION

The design of the key component of multisite scheduling environment, Distributed Scheduling Server (GDSS), is discussed in detail. Also, we focus on the multisite resource selection algorithm of GDSS. A three-level architecture of the Distributed Computational Grid Model (DCGM) is brought forward and acts as the infrastructure of multisite scheduling environment. A heuristic strategy, First – come First – serve (FCFS) algorithm, was described. In the FCFS algorithm, we mainly introduce the Multilevel Feedback Queue Scheduling (MFQS) algorithm to cluster the resources in the distributed computational grid. Meanwhile, we establish the applications of splitting the numbers depending upon the CPU Idle Rate and thus reducing the execution time shown in test bed.

The grid interfaces that will be used by the new schedulers, autonomic computing agents, and any number of other services yet to be developed for the grid. Open Grid Services Architecture is an open standard at the base of all of these future grid enhancements. It will make it easier to assemble the best products from various vendors, increasing the overall value of grid computing. Open Network project has produced tools that allow us to develop and test advanced scheduling techniques.

REFERENCES

1. L. Melloul, D. Beringer, N. Sample and G. Wiederhold, “CPAM, A Protocol for Software Composition," CAiSE'99, Heidelberg, Germany, June 1999.
2. Keith Swenson .Simple Workflow Access Protocol (SWAP),” IETF internet draft, August 1998”.
3. T. Pratt and M. Zelkowitz, ”Programming Languages, Design and Implementation”, 1996, Prentice Hall, Inc.
4. G. Wiederhold, D. Beringer, N. Sample, and L. Melloul, "Composition of Multi-site Services", Proceedings of IDPT'99, Kusadasi, Turkey, June 1999.
5. W. Rosenberry, D. Kenney and G. Fisher: “Understanding DCE”; O'Reilly, 1994.
6. C. Szyperski, “Component Software: Beyond Object-Oriented Programming”, Addison-Wesley and ACM-Press New York, 1997.

© JGRCS 2010, All Rights Reserved

7. G. Wiederhold, P. Wegner, and S. Ceri: "Towards Megaprogramming: A Paradigm for Component-Based Programming"; Communications of the ACM, 1992(11): p.89-99.
8. J. Hanly, E. Koffman and J. Horvath, Program Design for Engineers, Addison-Wesley, Menlo Park, CA, 1995.
9. T. Pratt and M. Zelkowitz, Programming Languages, Design and Implementation, 1996, Prentice Hall, Inc.
10. S.Bray, J. Paoli and C. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0," W3C Recommendation, February 1998.
11. R. Buyya, J. Giddy, D. Abramson, "A Case for Economy Grid Architecture for Service-Oriented Grid Computing," 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), April 2001.
12. P. Keyani, N. Sample, and G. Wiederhold, "Scheduling Under Uncertainty: Planning for the Ubiquitous Grid," Fifth International IEEE Conference on Coordination Models and Languages (Coord2002).
13. W. K. Shih, J. W. S. Liu, and J. Y. Chung. "Algorithms for scheduling imprecise computations with timing constraints," In Proc. IEEE Real-Time Systems Symposium, 1989.
14. M.J. Atallah et al, "Models and Algorithms for Co-scheduling Compute-Intensive Tasks on a Network of Workstations," Journal of Parallel and Distributed Computing, Vol. 16, 1992.
15. Grimshaw and W. Wulf. "Legion - a View from 50,000 Feet," Proc. 5th IEEE Symp. on High Performance Distributed Computing, pp. 89-99, IEEE Press.
16. P. Lawrence, editor, Workflow handbook 1997, John Wiley 1997.
17. T. Anderson, D. Culler, and D. Patterson, "A Case for Networks of Workstations: NOW," IEEE Micro, February 1995.

AUTHORS BIOGRAPHY

Mr. G. Mohammed Nazer obtained his MCA and M.Phil. in 1992 and 2009 respectively. He is currently Head, Department of Computer Applications at IFET College of Engineering, Villupuram. He has 6 years of Industrial experience as Software Engineer in Switzerland and 6.5 years of teaching experience. He is currently pursuing Ph.D. in Computer Science. His interesting research area is "Network Security, Security Controls and Cryptography". He has published one paper in International Journal.

Dr. S. S. Jayachandran has obtained his B.E(MechanicalEngineering) from Thiagarajar College of Engineering in 1969, Diploma in Statistical Quality Control and operations Research from Indian Statistical Institute, Chennai in 1974, M.E (Industrial Engineering) from College of Engg., Guindy, Chennai in 1974. He completed his Ph.D. in 1993 from Anna University, Chennai. He has carried out projects in the areas of Man power development under CPS for Ministry of HRD, Vocational training scheme for TAHDCO and Infrastructural Development Scheme for DRDA. He has carried out developmental works for over 100 industries and has conducted a number of Staff & Student development training programs. He has to his credit 66 technical publications and guided over 800 students. He is currently Principal, IFET College of Engineering, Villupuram.

© JGRCS 2010, All Rights Reserved