CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

A distributed system is one which provides a single system view to the collection of machines. Independent and physically scattered computing resources falling under the roof of a single large computer is called grid.

Grid computing can be generally defined as the process of integrating various technologies to provide better solutions. This forms a part of distributed computing. The computing resources may be geographically separated and is heterogeneous. Heterogeneity can be in the architecture of the nodes used or the operating system it uses.

For the resources to communicate, the components of the grid must lie on a network. The basic difference between a network and distributed systems is that, a network just provides a pipe for the resources to communicate whereas in the case of distributed systems the communication between the nodes take place in a more coordinated fashion.

Generally open standards and protocols are used as they provide mechanisms needed to create communications between components developed by different vendors. They also allow the developers to concentrate on the business logic rather than on the method of programming communication routines.
Further there should be secured and coordinated access to the grid. Secured access is controlling the users who can use the grid. This can be done by maintaining access control lists. Coordinated access is the idea of a scheduler for jobs and processes. Implementing features to provide quality of service (QOS) comes with coordination.

1.2 CLUSTER AND GRID COMPUTING

Cluster computing is the technique of linking two or more computers into a network (usually through a local area network) in order to take advantage of the parallel processing power of those computers. They are homogeneous in nature.

Grid computing, most simply stated is distributed computing taken to the next evolutionary level (Buyya 1999). 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.

1.2.1 Effective Utility of Underutilized Resources

Most of the computer systems connected in a network are found to be idling in the desktop of many organizations. The resources both software and hardware are not utilized in an effective manner. The advantage of cluster computing system is that unutilized CPU cycles can be used for running an application or for solving a problem. The easiest use of cluster 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 cluster.
1.2.2 Parallel CPU Capacity

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the biomedical field, financial modeling, oil exploration, motion picture animation, and many others. The common attribute among such uses is that the applications can be partitioned into independently running parts. A CPU intensive grid application can be decomposed into smaller jobs and each job is distributed to different machines for computation. A perfectly scalable application will, for example, finish 10 times faster if it uses 10 times the number of processors.

1.3 TYPES OF GRID

Based on the nature, computation grid can be classified as follows.

1.3.1 Computational Grid

This type of model allows resources to donate computing power to the grid whenever the workload demands. This is suitable for applications requiring very high processing power which a single system can’t provide or at a high cost. Such a grid takes advantage of the idle computational cycles in an organization and uses them for the needy applications. This is generally a client-server model. This type of grid allows the reuse of the infrastructure and client application for more projects at the same time. In this kind we cannot guarantee the time needed to get the result. Hence it is intended for applications which are not time critical.
1.3.2 Data Grid

Data Grid allows the data consumers to see a unified image of the respective information or data spread across different resources, potentially based on different technologies.

1.3.3 Network Grid

This type of grid tends to use the bandwidth. Every machine, servers and desktops in an organization has underused network bandwidth, which can be considered as an idle resource. When a given user or machine requires more resources from the network, a bottleneck is reached. In order to provide solutions to the bottleneck the bandwidth can be effectively used.

1.3.4 Multipurpose Grid

This is perhaps the more common implementation in the future of grid computing. The infrastructure of this grid should be adaptive enough to provide any of the grid models.

1.3.5 National Grid

National grids can be hosted by country. Such grids are useful in emergency situations, such as earthquakes, terrorist attacks etc. National grids can also support scientific investigations, such as studies of climatic changes, space station design and environmental cleanup.

1.3.6 Project Grid

Project grids are created to work on a specific goal. They are typically constructed from shared resources for a limited time and are designed to meet the needs of multi-institutional research groups and "virtual teams".
1.3.7 **Private Grid**

Private grids are sometimes called local grids or intra-grids, and are used by institutions such as hospitals and corporations. These grids are relatively small and centrally managed.

1.3.8 **Peer To Peer Grid**

**Peer-to-peer grids** rely on a "give to get" philosophy, where users exchange data with other users. The now defunct Napster is an example of old peer-to-peer technology. However, newer versions are truly peer-to-peer, requiring no third-party intervention.

1.3.9 **Cloud Like Grid**

Cloud like grids allow users to rent extra computing power by paying the computer owners. Examples are services offered by Amazon or Google.

1.4 **ROLE OF GRID USERS**

- Enrolling and installing grid software
- Logging onto the grid
- Queries and submitting jobs
- Data configuration
- Monitoring progress and recovery
- Reserving resources
1.5 MESSAGE PASSING INTERFACE (MPI)

One of the basic methods of programming for parallel computing is the use of message passing libraries (Bo Song et al 2007). These libraries manage transfer of data between instances of a parallel program running (usually) on multiple processors in a parallel computing architecture.

1.6 PARALLEL ARCHITECTURE

Parallel computers have two basic architectures: *distributed memory* and *shared memory*.

1.6.1 Distributed Memory

Parallel computers are essentially a collection of serial computers (nodes) working together to solve a problem. Each node has rapid access to its own local memory and access to the memory of other nodes via some sort of communications network, usually a proprietary high-speed communications network. Data are exchanged between nodes as messages over the network.

1.6.2 Shared Memory

In shared memory computer, multiple processor units share access to a global memory space via a high-speed memory bus. This global memory space allows the processors to efficiently exchange or share access to data. Typically, the number of processors used in shared memory architectures is limited to only a handful (2 - 16) of processors. This is because the amount of data that can be processed is limited by the bandwidth of the memory bus connecting the processors.

The latest generation of parallel computers now uses mixed shared/distributed memory architecture. Each node consists of a group of 2 to
The first step in designing a parallel algorithm is to decompose the problem into smaller problems. Then, the smaller problems are assigned to processors to work on simultaneously. Roughly speaking, there are two kinds of decompositions.

- **Domain decomposition**
- **Functional decomposition**

### 1.7.1 Domain Decomposition

In domain decomposition or "data parallelism", data are divided into pieces of approximately the same size and then mapped to different processors. Each processor then works only on the portion of the data that is assigned to it. Of course, the processes may need to communicate periodically in order to exchange data.

Data parallelism provides the advantage of maintaining a single flow of control. A data parallel algorithm consists of a sequence of elementary instructions applied to the data: An instruction is initiated only if the previous instruction is ended. Single-Program-Multiple-Data follows this model where the code is identical on all processors.

### 1.7.2 Functional Decomposition

Frequently, the domain decomposition strategy turns out not to be the most efficient algorithm for a parallel program. This is the case when the pieces of data assigned to the different processes require greatly different
lengths of time to process. The performance of the code is then limited by the speed of the slowest process. The remaining idle processes do no useful work. In this case, functional decomposition or "task parallelism" makes more sense than domain decomposition. In task parallelism, the problem is decomposed into a large number of smaller tasks and then, the tasks are assigned to the processors as they become available. Processors that finish quickly are simply assigned more work.

Task parallelism is implemented in a client-server environment. The tasks are allocated to a group of slave processes by a master process that may also perform some of the tasks. The client-server paradigm can be implemented at virtually any level in a program. For example, if you simply wish to run a program with multiple inputs, a parallel client-server implementation might just run multiple copies of the code serially with the server assigning the different inputs to each client process. As each processor finishes its task, it is assigned a new input. Alternately, task parallelism can be implemented at a deeper level within the code.

1.8 USE OF JAVA IN CLUSTER COMPUTING

Java (Mark.A.Baker et al 2006) is the best suited language to develop a clustering tool. Java programs are executed by a Java Virtual Machine (JVM), which interprets intermediate compiled byte code that is nominally platform independent. Although early versions of Java interpreted unoptimized byte code in a relatively unsophisticated manner, recent developments including static analysis, just-in-time compilation, JVM optimization, and instruction-level optimizations have improved execution efficiency. Consequently, Java is now competitive with C and C++ for few applications and on some platforms. The suitability of using Java in developing a clustering tool is quiet explorabale. The java functionalities can be effectively used to shape the tool in a flexible manner.
Java in its current state is efficient for high-performance applications. The advancement in virtual machine optimization has improved the performance of Java applications. Ghahramani et al. (2006) discuss the suitability of Java for high-performance applications. Java Grande Forum is formed to improve Java for Grande and high performance applications.

One advantage of the use of Java Platform in cluster computing is the security architecture implemented in the Java Virtual Machine (JVM). The sandbox security architecture provides mechanism to protect a machine from malicious actions from a foreign program. The security architecture is also flexible for a policy maker to strike the balance between resource protections and availability.

1.9 ROLE OF JAVA RMI

The Java RMI (Guillermo.L.Teboada et al 2007) supports well and the tool developed using JavaRMI helps in choosing the policies according to the application and the user need. Best node can also be selected to submit the job based on the current load. As modularity is available any module can be modified, enhanced and plug in again to make it work in a most effective way. Java Developer Kit (JDK) is utilized to develop an effective Cluster tool for various computationally complex applications.

Due to the growing need of the Internet and networking technologies, simple, powerful, easily maintainable distributed applications need to be developed. These kinds of applications can benefit greatly from distributed computing concepts. Java distributed computing technology has given developers a new solution to solve critical distributed application problems. JDK software and Java remote method invocation (Java RMI) technology provides a consistent programming platform that significantly simplifies distributed application development.
Remote Method Invocation enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. The Java remote method invocation (RMI) system allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another Java Virtual Machine. RMI provides for remote communication between programs written in Java programming language.

Remote Method Invocation applications are often composed of two separate programs, a server and a client. A typical server application creates some remote objects, makes references to them accessible, and waits for clients to invoke methods on these remote objects. A typical client application gets a remote reference to one or more remote objects in the server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth.

One of the most significant capabilities of the Java platform is the ability to dynamically download Java software from any Uniform Resource Locator (URL) to a Java virtual machine (JVM) running in a separate process, usually on a different physical system. The result is that a remote system can run a program, for example an applet, which has never been installed on its disk.

1.9.1 Working Principle of RMI

The server must first bind its name (specified by the user or programmer) to the registry that can be available to the clients for servicing the requests. At first, for each client, it creates a proxy (skeleton) object and by using that the clients can communicate with the server. For each client request, a server thread is created and launched. So the client can invoke more number of invocations. The server responds to all clients requests.
The client lookup the server by using the system name and server name (as binded in server side registry) to establish remote references. After establishing connection with the client the server communication is feasible.

**Figure 1.1 Working principle of RMI**

### 1.9.2 Stub And Skeleton

Whenever a client invokes a remote method, the call is first forwarded to stub, which is responsible for contacting the remote object. The stub is responsible for sending the remote call to the server-side skeleton. The stub opening a socket to the remote server, marshalling the object parameters and forwarding the data stream to the skeleton. A skeleton contains a method that receives the remote calls, demarshalls the parameters, and invokes the actual remote object implementation.
The Stub serializing the parameters to skeleton, which means that it converts the data (user defined type) into the format that can be transmitted over the communication channel (Marshalling). Likewise, at the server-side, the skeleton demarshall the request and make the actual method call. After the remote method gets executed, it starts serializing the produced result and sends it back to the stub which resides at the client. The stub then demarshalls that reply and gives the result arrived to the caller. Nowadays there is no separate stub and skeleton classes. Only stub class is created for the remote object and resides at both server and the client.

1.9.3 Steps for Developing RMI System

- Define the remote interface
- Develop the remote object by implementing the remote interface.
- Develop the client program.
- Compile the Java source files.
- Generate the client stubs and server skeletons.
- Start the RMI registry.
- Start the remote server objects.
- Run the client
1.10 SCOPE OF THE THESIS

Since the advent of computing there have been certain classes of problems that are so computational intensive and the solution may demand parallel processing to solve them within a stipulated time. We should have a better platform to support parallel processing (Manish Parashar et al 1992). During the heyday of the super computer, access to hardware capable of performing parallel processing was limited and often expensive. Most of the complex high end applications even consume months together for producing result. Complex applications like Weather forecasting, Seismic Analysis, and Evolutionary Computational Process demand high computational power for running their programs.

The availability of powerful desktop computers and high-speed connectivity has lead to distributed parallel processing environment (Schroeder 1993). Therefore, distributed computing is a method by which smaller institutions lacking financial resources can harness the equivalent computing power of a multi-processing supercomputer. The clustering algorithm directly affects the performance of the entire system (Stephen S. Yau and Haiqing Ying 1995). Care should be taken to know about the configuration of the computer and the effective method of sharing the load among computers. These are the two factors that determine the overall efficiency of the cluster (William E. Biles 1999).

Many tools (Adambeguelin et al 1993) are evolved for performing this Distributed computing. Utilizing the available resources in any organization or institution helps them to meet out the demand in computational power by complex applications. Parallel Virtual Machine (PVM) is one of the popular systems specifically designed to allow heterogeneous collection of computers hooked together by a network to be
used as single large parallel computer. PVM and Message Passing Interface (MPI) utilize the message passing libraries to perform the computational processes. The basics comparison of PVM and MPI (Dennis Guster 2003) reveals that MPI program is better than PVM.

Apple Macintosh introduced parallel operation control heuristic application (POOCH) (Dean E. Dauger and Viktor K. Decyk 2005) and Xgrid (Kremer and MacInnis 2006) that helps for distributing the job, running the job and collecting the result by which it takes less time than it would as a single system for higher throughput computing.

The Globus tool kit (Foster 2006) is a widely accepted tool for building a computational grid. The installation of server and client requires several important tools and the creation of computational environment using Globus tool kit is really challenging job to the grid users. The number of installations and the integration of those components are not easy tasks.

There is a need for grid users to customize their distributed computing environment according to the applications. Choosing the best node policies in the computational environment and the scheduling policies that suit their applications. Few tools like SUN Grid Engine (Babu Sundaram et al 2006) also demand the applications to be submitted in shell script.

Java is a best suited language to develop a clustering tool (Mark A Baker et al 2006). Java programmes are executed by a Java Virtual Machine (JVM) which interprets intermediate compiled byte code that nominally platform independent. The Java RMI (Guillermo. L. Taboada et al 2007) supports well to develop a tool that helps the user to choose the policies (Best Node, Scheduling) according to their application.
The Novel Job Distribution tool developed in this thesis has five parts. The first part deals with design of User GUI that provides enough useful information regarding the status of the job submitted. It helps the user to know whether the job is in the Waiting state, Running state or in Execution state thereby the user can plan his submission process accordingly. The second part is on designing the scheduler that plays a significant role in the computational cluster. This segment takes the sole responsibility of scheduling the tasks to the various nodes. By incorporating intelligence to this scheduler, effectiveness of the computational process can be improved. The third part deals with the Queue Manager which maintains the Queue. The effective management of the queue helps in improving the computational speed of the job execution as well as minimizes the job’s waiting time in the queue. The fourth part is the Grid Client basically called as Remote Manager which has an RMI Registry and supports in processing the specific task assigned to it and the final part is the design of Parallel job Manager which really helps in improving the computational speed of a task by making them parallelly executable. The job when it gets decomposed into smaller tasks, get assigned to various processors for parallel execution. The intelligence lies in choosing the least loaded node for assigning the job and by choosing effective scheduling policies, in order to achieve the required computational speed.

In a nutshell, the intention of the study is to analyze various Cluster and Grid Computing tools and to implement a novel distribution tool that has better features than the existing tools.

The intention of the present work is to,

- Provide a flexible design of grid management software to add or remove components.
- Re-submit the jobs when the execution (or) submitted node crashes.
• Allocate jobs dynamically to the execution nodes based on their loads.
• Free the user from submitting jobs through shell script.
• Provide options for choosing the scheduling policies and the best node.
• Obtain maximum throughput

1.11 ORGANISATION OF THE THESIS

This thesis presents a novel job distribution tool for an effective computational cluster. Chapter 1 gives an introduction to Cluster and Grid Computing and Java Remote Method Invocation. Chapter 2 presents the details of the literature review made. Chapter 3 presents details about the existing distributed computing tools. Chapter 4 describes the Development of New Job Distribution Tool using Java RMI. Chapter 5 discusses the implementation and Evaluation of Novel Job Distribution tool that includes modules like User GUI, Job scheduler, Queue Manager, Grid Client and Parallel job Manager. Chapter 6 provides the conclusion and future enhancements.