CHAPTER 3

EXISTING DISTRIBUTED COMPUTING TOOLS

3.1 PREAMBLE

Today’s emerging technologies have paved ways and means to develop cluster and grid computing tools. Every organization tries to have a tailor made cluster and grid computing tool that satisfies their need. This chapter deals with various cluster and Grid computing tools developed in various periods which helps in developing new job distribution tool. Tools like Parallel Virtual Machine (PVM), Parallel Operation Control Heuristic application (Pooch), Xgrid, Globus Tool Kit (GT) and SUN Grid Engine (SGE) are discussed.

3.2 PARALLEL VIRTUAL MACHINE

Distributed computing is accomplished using parallel processing in a loosely coupled environment. Parallel processing breaks large tasks into smaller ones for distribution among several computers. The computers communicate with each other through various programming implementations. Two such methods are Parallel Virtual Machine (PVM) and Message Passing Interface (MPI).

PVM is one of the first software systems specifically designed to allow heterogeneous collection of computers hooked together by a network to be used as a single large parallel computer (Beaumont et al 2003). The hardware in a user’s virtual machine may be single processor workstations, vector machines or parallel supercomputers or any combination of those. The
individual elements may all be of same type (homogeneous) or all different types (heterogeneous) or any combinations and all of them are connected through one or more networks.

These networks may be as small as LAN connecting machines in the same room or as large as the internet connecting machines across the world. This ability to bring together diverse resources under a central control allows the PVM user to divide a problem into subtasks and assign each one to be executed on the processor architecture that is best suited for that subtask.

![Parallel Virtual Machine: PVM 3.4](parallel_virtual_machine.png)

**Figure 3.1 Parallel Virtual Machine tool**

### 3.2.1 Parallel Program Paradigm

The master_slave model is used in which a separate control program termed the master is responsible for process spawning, initialization, Collection and display of results and perhaps the timing of the functionaries. The slave programs perform the actual computation involved.
### 3.2.2 Sample code:

For $i:=0$ to Num Workers-1

Pvm_spawn(<worker name>) Startup worker $i$

Pvm_send (<worker tid >,999) Send task to worker $i$

End for

Receive_send

While(work to do)

Pvm_recv(888) Receive result

Pvm_send(<available worker tid>,999) Send next task to available worker

Display result

End while.

Gather remaining results

For $i:=0$ to Num Workers-1

Pvm_recv(888) Receive result

Pvm_kill (<worker tid i>) Terminate worker $i$

Display result

End for.

Work load allocation is carried out by either data decomposition or functional decomposition.

### 3.3 COMPONENTS OF PVM

#### 3.3.1 Task Identifier (tid)

PVM uses a task identifier (tid) to address pvmds, task and groups of tasks within a virtual machine.
3.3.2 **PVM daemon**

One pvmd runs on each hosts of a virtual machine. Pvmcs owned by one user does not interactive with those owned by others. The pvmd serves as a message router and controller. The first pvmd is designated as master, while the others are called slaves. During normal operation, all are considered equal, but only the master can start new slaves and add them to the configuration. Reconfiguration requires originality on a slave host are forwarded to the master. Likewise only the master can forcibly delete host from the machine. The Shadow pvmd runs as the master host and is used by the master to start new slave pvmds.

3.3.3 **Programming Library:**

The libpvm library allows a task to interface with pvmd and other tasks. It contains functions for packing and unpacking messages and functions to perform pvm syscalls by using the message function to send service request to the pvmd.

3.3.4 **Starting slave pvmd:**

Two widely available systems are rsh and rexec(). rsh is a privileged program that can be used to run commands on another host without a password, provided the destination host can be made to trust the source host. 

Pvm_addhost() can be used to add new host. A resource manager is a pvm task responsible for making task and host scheduling decisions. The number of Resource Managers registered can be vary form one for an entire virtual machine to one per pvmd. The resource manger running on the master host manages any slave pvmds that don’t have their own resource managers. Pvm communication is based on TCP, UDP and UNIX domain sockets.
### 3.3.5 Computer System Specifications

**Master node**
- PIV 1.6 Ghz
- 256 MB RAM
- HDD 40 GB
- NIC 10/100
- Intel pro

**Slave node 1**
- PIV 2.4 Ghz
- 512 MB RAM
- HDD 40 GB
- NIC 10/100
- Intel pro

**Slave node 2**
- Pentium pro 200 Mhz
- 64 MB RAM
- HDD 40 GB
- NIC 10/100
- Intel pro

The program considered here is helloworld.c which is submitted to the Master node. The performance of the systems are studied and tabulated.

### 3.3.6 Sample programming code

```c
#include <stdio.h>
#include "pvm3.h"
```
main()
{
    cc = pvm_spawn("hello_other", 0, 0, ",", 1, &tid);
    if (cc == 1) {
        cc = pvm_recv(-1, -1);
        pvm_bufinfo(cc, 0, 0, &tid);
        pvm_upkstr(buf);
        printf("from t%x: %s\n", tid, buf);
    } else ... 
}
#include "pvm3.h"

main()
{
    int ptid;
    char buf[100];
    ptid = pvm_parent();
    strcpy(buf, "hello, world from ");
    gethostname(buf + strlen(buf), 64);
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, 1);
    pvm_exit();
    exit(0);
}
Table 3.1 PVM hosts and their execution speed

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Computer System ID</th>
<th>Execution time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PVM_Host_1</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>PVM_Host_2</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>PVM_Host_3</td>
<td>20</td>
</tr>
</tbody>
</table>

Figure 3.1 shows the PVM tool downloaded for testing purpose. Two major components of PVM are the pvmd (PVM daemon) and the PVM libraries. Table 3.1 shows the execution speed that minimizes when number of processors or nodes are increased in the computational environment.

3.3.7 Comparison of PVM and MPI

The basic comparison of PVM and MPI reveals that MPI performs better than PVM (Dennis Guster 2003). After prior experimentation, Dennis Guster has concluded in his paper that PVM’s processing time in seconds is not much better than the MPI’s processing time in seconds and the tabulation is as shown below, but MPI is object oriented, and capable of managing the available resources (Skjellum and Antony 1998). PVM is not capable of independently managing resources and relies on applications such as Condor, DQS(Distributed Queuing system), or Load Leveler that are "PVM aware" to perform job scheduling and process management.
Table 3.2 PVM versus MPI

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>PVM (Processing Time in Seconds)</th>
<th>MPI (Processing Time in Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>57×57</td>
<td>0.140035</td>
<td>0.065717</td>
</tr>
<tr>
<td>75×75</td>
<td>0.217412</td>
<td>0.133779</td>
</tr>
<tr>
<td>99×99</td>
<td>0.269811</td>
<td>0.215235</td>
</tr>
<tr>
<td>165×165</td>
<td>1.127777</td>
<td>1.037093</td>
</tr>
<tr>
<td>255×255</td>
<td>3.28437</td>
<td>2.025409</td>
</tr>
</tbody>
</table>

Table 3.2 clearly reveals that MPI consumes lesser processing time than PVM. Hence MPI is considered in developing a novel distribution tool.

3.4 POOCH

POOCH provides a Macintosh user interface for distributing and initiating a numerically-intensive parallel application on a networked cluster of Macintoshes (Dean E. Dauger and Viktor K. Decy 2005). It coordinates the distribution of data, establishing communication among pooch components installed in various machines and provides a user interface to start and monitor parallel computing jobs. Pooch technology allows the user to combine supercomputing-level features with an easy-to-use modern user interface.

POOCH is an unique technology design that allows flexibility to operate among wide variety of network environments and cluster configurations. It supports the industry-standard Message-Passing Interface, making it easy to write portable supercomputing code on a Mac cluster. At the same time it supports grid and distributed-computing models.
3.4.1 Experimental Methodologies

Developing an experimental methodology for cluster computing environment involves:

i) Establishing physical connection between computer systems to form a cluster. This is accomplished by connecting built-in Ethernet card equipped in every eMac to a switch through an individual Input / output box using UTP category 5E type cable crimped with RJ45 connector.

ii) Installation of POOCH on each system in the cluster environment.

iii) This is done by double clicking the POOCH software.

iv) Installation of MPICH on each system that participate in the cluster.

v) This is performed by double clicking the MPICH software.

vi) Development of MPI implementation with C.

Following issues should be covered while designing a program in MPI with C

i) Available hardware platforms are identified.

ii) Parallelism in the application are identified (ptrap.c)

iii) The mpi implementation through POOCH is executed.

iv) The execution speed is noted and tabulated.

3.4.2 Experimental Cluster Configurations

Cluster is formed with the available nodes connected in the LAN environment. Selective nodes are made to participate in the cluster. The eMac machines can be added by the Add option radio button available. If we click the button, the machine will be added in the cluster otherwise it won’t be
added. In Figure 3.2 it shows the nodes selected or added to participate in a cluster. The nodes not selected or not added will not participate in the cluster and hence the job will not be submitted to those nodes though they are available in the network.

![Node Scan Window](image)

**Figure 3.2** Nodes selected (Added) to participate in the cluster

Even though the computer nodes have different versions of same OS and hardware configurations, for simplicity, Computer nodes with similar hardware configurations are considered as given below:

- Processor : 700 MHz eMac G4
- Memory : 128 MB
- L2 Cache size : 256 K
- Machine model : eMac (version=2.1)
- Software overview of cluster nodes
  - System version : Mac OS X 10.2.8(6R73)
  - Kernel version : Darwin Kernal version
3.4.3 Evaluation of POOCH Tool

The POOCH family includes apple’s Mac OS X computer systems to form a cluster.

![Diagram of POOCH architecture](image)

**Figure 3.3 The Architecture of POOCH**

Figure 3.3 shows the submission of a job to the controller by a client and subsequently how the job gets split into smaller tasks by the controller and distributed to various other nodes for computation. The results are collected back by the controller from different agents and returned to the client who submitted the job.
Figure 3.4 Opening the POOCH application version 1.5.5

The POOCH application is a built in tool provided by Apple Macintosh systems. The above Figure 3.4 shows the opening of POOCH application. This tool supports in creating a homogeneous cluster. Figure 3.5 shows the nodes that get added onto the cluster environment. The applications (ptrap.c is the application considered here for getting executed) can also be chosen for execution. The apple Macintosh machines having an option of not to participate in the cluster. In the cluster computing environment any node can be designated as controller. The agents may be of dedicated, distributed or wireless type. POOCH provides an opportunity for the user to know about the application submitted details.

The Controller takes care of connection establishment and passing messages among the nodes. Figure 3.6 shows how the controller receives the confirmation reply from the agent. The client will contact the controller to receive its output.
Figure 3.5 Nodes and application selection to run in POOCH

Figure 3.6 Controller receives confirmation reply from the agent
Figure 3.7 Execution results on three nodes

The controller decomposes the job and distributes them to the relevant nodes. The consumption of execution of time is found to be more. When more number of nodes is participating in the cluster environment, the submitted application is executed at a faster rate. Figure 3.7 shows the results of running an application shared among three nodes. Table 3.3 shows the results of number of nodes involved in computation and their execution speed.

Table 3.3 Nodes used and their execution speed

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Computer system id</th>
<th>Execution time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>emac 41</td>
<td>0.01503</td>
</tr>
<tr>
<td>2</td>
<td>emac 41, emac 46</td>
<td>0.00568</td>
</tr>
<tr>
<td>3</td>
<td>emac 41, emac 46, emac 48</td>
<td>0.002351</td>
</tr>
</tbody>
</table>
When more number of nodes is increased the execution time considerably decreases. Figure 3.8 shows the execution time and the number of machines. Other OS are not supported by this tool. The scalability is also an issue where we cannot have more machines in the environment.

![Performance Evaluation](image)

**Figure 3.8** Execution time versus number of machines

### 3.4.4 Results

This section presents experimental results of an application run on a parallel implementation. Figure 3.9 shows the variation in execution time for running the same job for different cluster configurations. It clearly shows a reduction in execution time for increase in the number of nodes in the cluster configurations.

### 3.5 XGRID

XGRID (Kremer D. Maclnnis 2006) is a technology from Apple which takes long-running computation tasks, makes it easy to distribute to a
centrally managed cluster of Macintosh computers. The computer systems connected in the cluster executes the deployed application in parallel. The execution time becomes lesser when the same application runs on a shared cluster environment. The architecture has three components namely client, controller and agent.

3.5.1 Architecture

Xgrid’s architecture is similar to other desktop middleware systems such as Condor, and consists of the:

i) Agents

ii) Client

iii) Controller

The architecture of the XGRID system is designed around a job based system; the controller sends agents jobs, and the agents return the responses. The actual computation that the controller executes in an XGRID system is known as a job.

The job contains all the files required to complete the tasks successfully such as the input parameters, data files, directories, executables and/or shell scripts, the files included in an XGRID job must be able to be executed either simultaneously or asynchronously, or any benefits of running such a job on an XGRID is lost. Once the job completes, the controller can be set to notify the client of the task’s completion or failure. The client can leave the network while the tasks are running. It can also monitor the job status on demand by querying the controller, although it cannot track the on going progress of individual task.
**Client:** The client is a command line tool running within the terminal application. It contacts the controller and is used to describe, submit, monitor and retrieve jobs submitted to the cluster.

**Controller:** The Xgrid controller manages the communication between clients and agents. It accepts connections from both entities. None of them can directly connect to each other. It receives job submissions from clients, breaks the jobs into tasks, and, if needed, stores them in a waiting queue and dispatches tasks to available agents. Once agents return results it forwards them back to the clients.

**Agents:** The Xgrid agent executes the tasks simultaneously and returns results to controller

The controller plays a vital role in an XGRID, as this node is responsible for the distribution, Supervision and coordination of tasks on agents. The program running on the controller can assign and reassign tasks to handle individual agent failures on demand.

For the purpose of testing mac clusters, the altivecfractal carbon demo, a demonstration parallel application, is available. This demonstration of high performance computing also runs on a single node. The user runs this application in parallel by selecting new job from the file menu. This action opens the newjob window. The user may drag the altivec fractal carbon demo from the finder to this job window. To setup a parallel computing job, the altivec factor carbon demo is dropped in the job window. Next, the user chooses nodes to run in parallel. By default, XGRID selects the node where the job is being specified. To add more, the user clicks on select node, which invokes a node scan window. Finally the parallel job must be started by clicking on launch job. Now the copies of the parallel application to other nodes are sent and the execution is initiated in parallel. Upon completion of
its computational task, the performance is tabulated which is significantly greater than single node performance.

### 3.5.2 XGRID Job Management

If any job or task fails, the scheduler will automatically resubmit it to the next available agent. The controller will cache all the required data, manage scheduling and task-level failover, then hold and return all the results, including standard output and error from each task. If requested, the controller will notify the user via email that the job has been completed. The user can then retrieve the results from any authenticated Xgrid client system using the relevant job ID.

The only limitation is that no agent can be moved or sub-pool deleted as long as jobs within it are running. The Jobs button displays the list of all jobs managed by each controller or sub-pools on a controller. Each job can be individually stopped, restarted or removed. The current status of each job is also displayed with the actual CPU power used, the job progression and the number of sub-tasks belonging to this job.

It is useful to note that an Xgrid client is detachable from the network even while jobs are being executed (Hughes and Baden 2006). Even when the client is disconnected from the grid due to network failure, jobs submitted by him on earlier time can be completed and the results may be retrieved by the client from the controller once the network connectivity is reestablished. Xgrid allows for a range of agent types ranging from full-time agents, part-time (cycle stealing) agents, and remote agents (for distributed computational grids).

All communications between clients, controllers and agents are able to be strongly encrypted over the network ensuring in transit security across
segregated administrative domains. Xgrid software handles many tasks such as

i) Job queuing

ii) Job execution

iii) Node management

iv) Emphasis on ease of use

v) Rendezvous “zero configuration”

vi) Graphical cluster/job management interface

vii) Unix extendibility for power users

As administrator, you can stop, pause, resume or delete jobs or partition available agents into separate clusters, which allows different groups to share a single controller without interfering with each other. Figure 3.9 shows the XGRID software available in the desktop. It is built in software available within the system. Figure 3.10 clearly shows the number of machines participating in the cluster environment. Table 3.4 shows the number of systems chosen and their respective execution speed.

![Xgrid Technology Preview 2](image)

**Figure 3.9** XGRID software available at the desk top of the system
When the application is made to run on a single machine, the time consumed for execution is 40 seconds. When the same application is made to
run on two machines the execution speed is considerably increased and the time consumption for execution is reduced say 28 seconds. When the same application is deployed to three machines, the time consumption for execution is minimized say 25 seconds. Hence it is obvious that by increasing the number of nodes in the cluster environment the execution speed of an application is considerable increased.

3.6 GLOBUS TOOL KIT VERSION 4 (GT4)

The Globus toolkit (Foster 2006) is a widely accepted tool for building computational grid. The installation of the server and client requires several important tools and the creation of computational environment. The number of installations and the integration of components is also a great task. Configuration of GT4 is carried out based on the user requirement. The numbers of clients depend on the size and running time of the application. For experimental purpose a file is searched using the tool.

The file to be searched will be divided into modules and each such module will be searched in parallel, thereby distributing the load among several machines in order to reduce the total running time of the searching. The sequential running time and parallel running time of the search operation will be compared to prove that searching in parallel will consume much lesser time than running the searching in sequential.

3.6.1 The system overview of Globus Tool kit

The overview of Globus tool kit and the components involved in framing the computational environment is as shown in Figure 3.11.
3.6.2 Grid Resource Allocation Manager/Global Access to Secondary Storage

The primary components of the resource management pyramid are the Grid Resource Allocation Manager (GRAM) (Buyya et al 2000) and the Global Access to Secondary Storage (GASS).

3.6.3 MDS (GRIS/GIIS)

Based on the Lightweight Directory Access Protocol (LDAP) (Foster 2006), the Grid Resource Information Service (GRIS) and Grid Index Information Service (GIIS) components can be configured in a hierarchy to collect the information and distribute it. These two services are called the Monitoring and Discovery Service (MDS). The information collected can be

![Diagram of GT4 system overview]

Figure 3.11 System overview of GT4
static information about the machines, dynamic information showing current CPU or disk activity.

A rich set of information providers is included with the Toolkit and the Globus users can add their own. The information providers acts as an interface with the GRIS, which reports information such as resource availability, number of components etc to a hierarchy of GIIS servers in the grid. The LDAP query language is used to retrieve the desired information.

3.6.4 Grid File Transfer Protocol

Grid File Transfer Protocol is a key component for the secure and high-performance data transfer (Allcock et al 2005). The Globus Replica Catalogue and Management are used to register and manage both complete and partial copies of data sets. These three pyramids are modularized and can function in isolation however, together, they complement each other.

3.6.5 Grid Security Infrastructure

All of the above components are built on top of the underlying Grid Security Infrastructure (GSI). This provides security functions including single/mutual authentication, confidential communication, authorization, and delegation.

3.6.6 Grid Resource Allocation Manager

Grid Resource Allocation Manager (GRAM) (Feller et al 2000) is the module that provides the remote execution and status management of the execution. When a client submits a job, the request is sent to the remote host and handled by the gatekeeper daemon located in the remote host. Then the gatekeeper creates a job manager to start and monitor the job. When the job is
finished, the job manager sends the status information back to the client and terminates.

3.6.7 The Globusrun command

The Globusrun command submits and manages remote jobs and is used by almost all Grid Resource Allocation Manager Client tools. This command provides the following functions:

- Making a request for job submission to remote machines.
- Gets the standard output of job results from standard remote machines
- Uses GIIS to provide the secure file transfer among grid machines.

i) Apple macOS machines with OS Tiger 10.4 -2 No’s and Pentium 4 with Redhat linux as OS – 3 No’s

ii) To test the performance of the Globus Tool kit, the file test.c is considered.

iii) The files are not indexed

3.6.8 System Configuration

Apple machines: 2 no’s

- Processor : 700 MHz eMac G4
- Memory : 128 MB
- L2 Cache size : 256 K
- Machine model : eMac (version=2.1)
- Software overview of cluster nodes
- System version : Mac OS X 10.2.8(6R73)
- Kernel version : Darwin Kernal version
**Intel machines : 3 No’s**

- Processor : Pentium IV 2.4Ghz
- Memory : 512 MB
- L2 Cache size : 256 K
- Machine model : Intel pro
- Hard Disk Drive: 40 GB
- System version : Debian Linux 2.6.18

### 3.6.9 Parallel File Search

A file containing the location and names of all the files in the system is created by the system administrator. Every machine in the grid has the file. The user provides the file name to be searched as input to the program. The program in turn searches the entire file created by the administrator in parallel and output of the program will be the machine in which the file is present. Instead of searching all the machines sequentially which consumes more time the search operation is done in parallel in all the machines and hence saving time.

```c
#include <mpi.h>
#include <string.h>
#include <stdio.h>
#include <sys/timeb.h>
#define N 3
long int get_size(FILE *fp);
int main(int argc,char **argv)
```
{
    FILE *fp=NULL,*fp1=NULL,*fp2=NULL,*fp3=NULL;
    char ch,temp[1500];
    int rank,status,i=0;
    long int size;
    long int unit,end;
    double tstart,tend;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    fp=fopen("/usr/local/machine","r");
    if(rank==0)
        fp1=fopen("/usr/local/output1","w");
    if(rank==1)
        fp2=fopen("/usr/local/output2","w");
    if(rank==2)
        fp3=fopen("/usr/local/output3","w");
    tstart=MPI_Wtime();
    size=get_size(fp);
    unit=size/N;
    fseek(fp,rank*unit,SEEK_SET);
    if(rank==N-1)
        end=size;
    else
        end=rank*unit+unit;
}
while(ftell(fp)<=end) {
    ch=fgetc(fp);
    if(ch==EOF) break;
    if(ch!='\n') {
        temp[i]=ch;
        i++;
    } else {
        temp[i]='\0';
        i=0;
        if((strstr(temp,argv[1]))!=NULL) {
            if(rank==0)
                fprintf(fp1,"1: %s\n",temp);
            if(rank==1)
                fprintf(fp2,"2: %s\n",temp);
            if(rank==2)
                fprintf(fp3,"3: %s\n",temp);
        }
    }
    tend=MPI_Wtime();
    if(rank==N-1)
        printf("nFinish time:%lf seconds\n",tend-tstart);
MPI_Finalize();

long int get_size(FILE *fp)
{
    long int size;
    fseek(fp,0L,SEEK_END);
    size=ftell(fp);
    return size;
}

3.6.10 Snapshots

The File search result is as shown in the Figure 3.12.

![Figure 3.12 Output obtained after searching](image)

3.6.11 Experimental application

File search application is chosen for experimentation. The system administrator creates a file containing the location and names of all the files in
the system. Every machine in the grid has the copy of this metafile. Whenever the user interested in searching for an application file he provides the name of the file in the user window. The user provides the file name to be searched as input to the program. The system in turn makes a parallel search in all machines and identifies the machine which holds the file. The execution results of 1 to 5 systems are given in table 3.5.

3.6.12 Results

The Table 3.5 shows the execution time consumed for a parallel file search application. It is found that when a file is searched with the help of a single node, the execution time is found to be more whereas if the same file is searched using 2, 3, 4 or 5 systems the time consumed for finding the file becomes very short. Therefore if the number of nodes is increased in the cluster environment, the application deployed for execution consumes very less execution time.

Table 3.5 Execution with 1 to 5 systems in GT4

<table>
<thead>
<tr>
<th>File size / No of Machines</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 kb</td>
<td>0.000746</td>
<td>0.000186</td>
<td>0.000103</td>
<td>0.000190</td>
<td>0.000240</td>
</tr>
<tr>
<td>200 kb</td>
<td>0.000996</td>
<td>0.000306</td>
<td>0.000158</td>
<td>0.000160</td>
<td>0.000190</td>
</tr>
<tr>
<td>300 kb</td>
<td>0.001207</td>
<td>0.000501</td>
<td>0.000421</td>
<td>0.000240</td>
<td>0.000297</td>
</tr>
<tr>
<td>400 kb</td>
<td>0.001294</td>
<td>0.000632</td>
<td>0.000204</td>
<td>0.000210</td>
<td>0.000217</td>
</tr>
<tr>
<td>500 kb</td>
<td>0.001549</td>
<td>0.000932</td>
<td>0.000432</td>
<td>0.000326</td>
<td>0.000278</td>
</tr>
</tbody>
</table>

3.7 SUN GRID ENGINE

SUN Grid Engine (SGE) is a Distributed Resource Management (DRM) software supported by SUN Microsystems and used worldwide for
distributed resource management and software/hardware optimization in heterogeneous networked environments (Babu Sundaram et al 2006). It provides policy-based workload management and dynamic provisioning of application workloads. The SGE software enables to evolve resource management strategies to distribute jobs across any grid.

3.7.1 Components of SUN Grid Engine System

The Hardware components of the SUN Grid Engine are Hosts which are classified into the following four groups namely

- Master Host
- Execution Hosts
- Administration Hosts
- Submit Hosts

Grid engine system hosts are classified into four groups. Depending on which daemons are running on the system and on how the hosts are registered at `sge_qmaster`. The master host is central to the overall cluster activity. The master host runs the master daemon `sge_qmaster` and the scheduler daemon `sge_schedd`. Both daemons control all grid engine system components, such as queues and jobs. Execution hosts are systems that have permission to execute jobs. Therefore execution hosts have queue instances attached to them. Execution hosts run the execution daemon `sge_execd`. Administration hosts are hosts that have permission to carry out any kind of administrative activity for the grid engine system. In particular, a user who is logged in to a submit host can submit jobs, can monitor the job status. A system can act as more than one type of host.
3.7.2 Daemons

The important software components of sun grid Engine system are daemons, Queues, dispatching and LAM MPI.

Four daemons provide the functionality of the grid engine system.

- Master Daemon
- Execution Daemon
- Shadow Master Daemon
- Job Controller Daemon

Master Daemon maintains tables about hosts, queues, jobs, system load, and user permissions. The execution daemon is responsible for the queue instances on its host and for the running of jobs in these queue instances. The Shadow Master is a supporting daemon for Master Daemon when the Master Daemon is failed or crashed. Job Controller Daemon starts and terminates the job script and collects accounting information.

3.7.3 Queues

A queue is a container for a class of jobs that are allowed to run on one or more hosts concurrently. A queue determines certain job attributes, for example, whether the job can be migrated, throughout its lifetime, a running job is associated with its queue. Association with a queue affects some of the things that can happen to a job. For example, if a queue is suspended, all jobs associated with that queue are also suspended.

Jobs need not be submitted directly to a queue. You need to specify only the requirement profile of the job. A profile might include requirements such as memory, operating system, available software, and so forth. The grid
engine software automatically dispatches the job to a suitable queue and a suitable host with a light execution load. If you submit a job to a specified queue, the job is bound to this queue. As a result, the grid engine system daemons are unable to select a better-suited device or a device that has a lighter load.

3.7.4 Dispatching

Deciding a job's importance with respect to other pending jobs and running jobs, sensing the load on all machines in the cluster, and sending the job to a queue on a machine selected according to configured selection criteria.

3.7.5 LAM/MPI

The Message Passing Interface (MPI) is a set of API functions enabling programmers to write high performance parallel programs that pass messages between processes to make up an overall parallel job (Matsuda et al 2004). MPI is the culmination of decades of research in parallel computing, and was created by the MPI Forum – an open group representing a wide cross-section of industry and academic interests. LAM/MPI is not only a library that implements the mandated MPI API, but also the LAM run-time environment: a user-level, daemon-based run-time environment that provides many of the services required by MPI programs. Both major components of the LAM/MPI package are designed as component frameworks– extensible with small modules that are selectable (and configurable) at run-time. This component framework is known as the System Services Interface (SSI).
3.8 SUMMARY

The cluster and grid computing tools are discussed in this chapter. The Implementation and evaluation are also carried out to assess the performance of each and every tool. The evaluation of various tools shows that when an application is deployed in the master of the cluster, the execution time is found to be reduced when the number of nodes in the cluster environment is increased. The next chapter deals with the development of new job distribution tool and its architecture.

The Cluster and Grid Computing tools are very easy to handle. Most of the tools are found to be connected with user GUI and they are user friendly. Though there are variations found in the architectural setup, the function of every tool has the common objective of executing fast.

Every tool is experimented with varying number of nodes. Interesting fact is that the cluster mostly steals the CPU cycles of the machine and do not disturb any user of the machines.