

CPC

Cloud Parallel Computing Using UNIX OS



By

Capt Naeem Amjad

NC Imtiaz Kumar

NC Sreman Kumar

Supervisor:

Col Naveed Sarfaraz Khattak

Submitted to the Faculty of Computer Science

National University of Sciences and Technology, Rawalpindi in partial fulfillment for
the requirements of a B.E Degree in Computer Software Engineering

June 2013

CERTIFICATE

Certified that the contents and form of project report entitled “**Cloud Parrallel Computing Using UNIX OS**” submitted by 1) Naeem Amjad, 2) Imtiaz Qumar, and 3) Sreman Qamar have been found satisfactory for the requirement of the degree.

Supervisor: _____

Col Naveed Sarfaraz Khattak

ABSTRACT

Many Computation task are depending on heavy calculation and analyzing large amounts of data. These operations can take a long time on single computer. Cloud Parallel Computing utilizes networked computers by making them to work together to solve a problem, hence reducing the execution time.

“Cloud Parallel Computing Using UNIX OS” focuses on Implementation of Cloud Infrastructure from the perspective of providing parallel processing to the jobs submitted by users.

This system uses distributed memory model to perform network parallel computing. The communication among systems is done through Message Passing Interface (MPI). Our project will be running the parallel algorithms like matrix multiplication to display the notion of work efficiency. We are using JPPF on our system which is an open source Java message passing library that allows application developers to write and execute parallel applications for computer clusters/clouds

Upon submission of a job by the user, it is broken into discrete parts that are solved concurrently by further breaking them down to a series of instructions and executing each instruction simultaneously on different CPUs through Message passing Interface and end result of computation is displayed to user

This application proved to be very time saving and make the task of writing reports easier for administrators and free them for more productive tasks. Various Testing and evaluation results conducted on the product are extremely promising.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere.

DEDICATION

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents, without whose unflinching support and unstinting cooperation,

a work of this magnitude would not have been possible

ACKNOWLEDEMENTS

In the name of Allah, The Most Gracious, The Most Merciful, All the praises and thanks be to Allah, the Lord of the Universe, The Most Gracious, The Most Merciful. The only owner of the day of Recompense, You (Alone) we worship and You (Alone) we ask for help, Guide us to the straightway, The way of those on whom You have bestowed Your grace, Not of those who earned Your anger, Nor of those who went astray.

Nothing is achievable without the will of ALLAH. We are grateful to ALLAH, who has guide us and give us the strength to accomplish this task

We are also grateful to our parents for their unwavering faith in us, their continuous support and love without which we would not have been able to succeed.

We would specially like to thank our supervisor Col Naveed Sarfaraz Khattak from MCS who has been a great help for us in our project. We are highly thankful to all of our teachers and staff of MCS who supported and guided us throughout our course and research work. Their knowledge, guidance and training enabled us to carry out this research work.

In the end we would like to acknowledge the support provided by all our friends, colleagues, relatives and all the people who, in one way or the other had shared in our activities.

Table of Contents

Chapter 1	1
1. INTRODUCTION	1
1.1 Introduction	1
1.2 Background.....	1
1.3 Problem Statement	1
1.4 Objectives	2
1.5 Deliverables	2
1.6 Technological Requirements.....	2
2. LITERATURE REVIEW	4
2.1 Introduction	4
2.2 Literature Review	4
2.3 Previous Work.....	5
2.4 Issues Solved by CPC	5
2.5 Roles and Responsibilities	5
2.6 Work Breakdown Structure	6
2.7 Project Plan.....	6
3. SYSTEM REQUIREMENTS	7
3.1 Introduction	7
3.2 Product Scope	7
3.3 Product Perspective	8
3.4 Product Functions	8
3.5 User Classes and Characteristics	9
3.6 Operating Environment	9
3.7 Design and Implementation Constraints.....	9
3.8 User Documentation	9
3.9 Assumption and Dependencies.....	10
3.10 External Interface Requirements	10
3.10.1 User Interfaces.....	10
3.10.2 Hardware Interfaces.....	10
3.10.3 Software Interfaces	11
3.10.4 Communication Interface	11
3.11 System Features.....	11
3.11.1 Registration.....	12
3.11.2 Login	12
3.11.3 Submit Job	13
3.11.4 Job Initialization.....	14
3.11.5 View Job Status	14
3.11.6 Delete Job.....	15
3.11.7 Delete Node	Error! Bookmark not defined.
3.11.8 Add Node	16
3.11.9 Dispatch Job	16
3.12 Other Nonfunctional Requirements	17
3.12.1 Performance Requirements.....	17
3.12.2 Capacity	17

3.12.3	Security Requirements	17
3.13	Software Quality Attributes	18
3.13.1	Reliability:	18
3.13.2	Usability:	18
3.13.3	Maintainability	18
3.13.4	Testability:	18
3.13.5	Availability:	18
3.14	Other Requirements	19
4.	System Design	20
4.1	Introduction	20
4.2	Scope	20
4.3	Architecture Design	20
4.3.1	Architecture Pattern:	20
Architecture Pattern	22
4.4	Detailed Design	23
4.4.1	ER Diagram	23
Class Diagram:	Error! Bookmark not defined.
4.4.2	Use Cases:	24
4.4.3	Sequence Diagram	28
4.4.4	Activity Diagram:	30
5.	SYSTEM IMPLEMENTATION	32
5.1	Tools and Technologies	32
5.1.1	Apache ANT	32
5.1.2	Java Development Kit	32
5.1.3	PuTTY	32
5.1.4	Shell Scripting	32
5.1.5	WinScp	32
5.1.6	Secure Shell (SSH)	33
5.2	Software Implementation	33
5.2.1	Administration and Monitoring Console	33
5.2.2	Server Distribution	33
5.2.3	Node Distribution	33
5.2.4	Client Application	34
5.3	Implementation Terminologies and Code Snippets	34
5.3.1	Task Objects	34
5.3.2	JPPFTask	34
5.3.3	Job name and Identifier	36
5.3.4	Creating a job	37
5.3.5	Adding tasks to a job	38
5.3.6	Handling of execution results	38
5.3.7	Creating and closing a JPPFClient	39
5.3.8	Submitting a job	41
5.3.9	Cancelling a job	41
5.3.10	Receiving notifications for new and failed connections	42
6.	TESTING AND RESULT ANALYSIS	43
6.1	Testing	43
6.2	Results and Analysis	46
6.2.1	Results	46
6.2.2	Analysis	47
7.	CONCLUSION AND FUTURE WORK	48

List of Figures

Figures	Page Number
2.1 Literature Review	4
2.6 Work Breakdown Structure	6
4.3.1 Architectural Design	21
4.3.2 Implementation View	22
4.4.1 ER Diagram	23
4.4.2 Class Diagram	24
4.4.3 Use Case Diagram	25
4.4.4 Sequence Diagram	29
4.4.5 Activity Diagram	31

List of Tables

Tables	Page Number
2.5 Roles and Responsibilities	5
2.6 Project Plan	6
3.3 Operating System Requirements	9

INTRODUCTION

1.1 Introduction

This section is written to specify the related work/background of Cloud Parallel Computing Using Unix Operating System Project. Different streaming servers currently available to users are also included in the section. Moreover need of Cloud Parallel Computing is also justified.

This document also explains the objectives of Cloud Parallel Computing using Unix Operating System, deliverables of project (Cloud Parallel Computing Using Unix Operating System), technical requirements, and a comprehensive project plan, explaining the work break down among the respective group members.

1.2 Background

Cloud parallel computing is the simultaneous use of multiple computing resources (hardware and software) to solve a computation problem and these resources are delivered as a service over a network. Cloud Computing makes computer infrastructure and services available "on-need" basis. The computing infrastructure includes hard disk, development platform, database, computing power or complete software applications.

1.3 Problem Statement

Many Computation task needs lots of mathematical calculations and take longer time on a single computer. There is a need to establish a network of computer to perform such tasks by splitting them and performing on multiple computers and obtaining the result more efficiently. MCS Computer Software Engineering Department has sufficient no of Servers and network computer available to utilize for Cloud Parallel Computing.

1.4 Objectives

The objectives of our project include:

- i. To learn Cloud Parallel Computing Using Unix OS
- ii. To Understand the configuration and architecture of Ultra Sparc server with Unix operating Systems
- iii. To learn different techniques for Cloud parallel Computing
- iv. To develop an application for the end users to utilize the cloud parallel processing
- v. To implement cloud infrastructure from prospective of providing parallel processing to the jobs.
- vi. To configure a front end and a middleware using Sun Ultra Sparc workstation as server

1.5 Deliverables

Deliverables of the project are:

- i. Sun Ultra Sparc Server Configured as Master for Cloud Parallel Computing
- ii. Configured Nodes With Heterogeneous OS
- iii. Configured Administrative Console
- iv. Matrix Multiplication application implemented through Cloud Parallel Computing
- v. Documentation/User Manual

1.6 Technological Requirements

- i. Sun Ultra Sparc Server
- ii. Java Parallel Processing Framework
- iii. Networking Facility
- iv. Computer Lab with Heterogeneous OS
- v. MS Office for Documentation
- vi. Java Standard Edition Version 1.5 or Later

- vii. Unix Shell Scripting
- viii. SSH Client and Server
- ix. FTP Client and Server
- x. Apache Ant, Version 1.7.0 or Later

LITERATURE REVIEW

2.1 Introduction

This section is written to specify the related work/background of Cloud Parallel Computing Project and what has been done in the domain, what the issues are and what issues our project will resolve. Moreover need of need of Cloud Parallel Computing using Unix Operating System is also justified.

This document also explains a comprehensive project plan, explaining the work break down among the respective group members.

2.2 Literature Review

Presently parallel Cloud Computing Solutions are deployed on different platforms like Intel based Machines, Linux and Unix. Its purpose is to broke a problem into discrete parts in the form of series of instruction and to run on multiple CPUs simultaneously

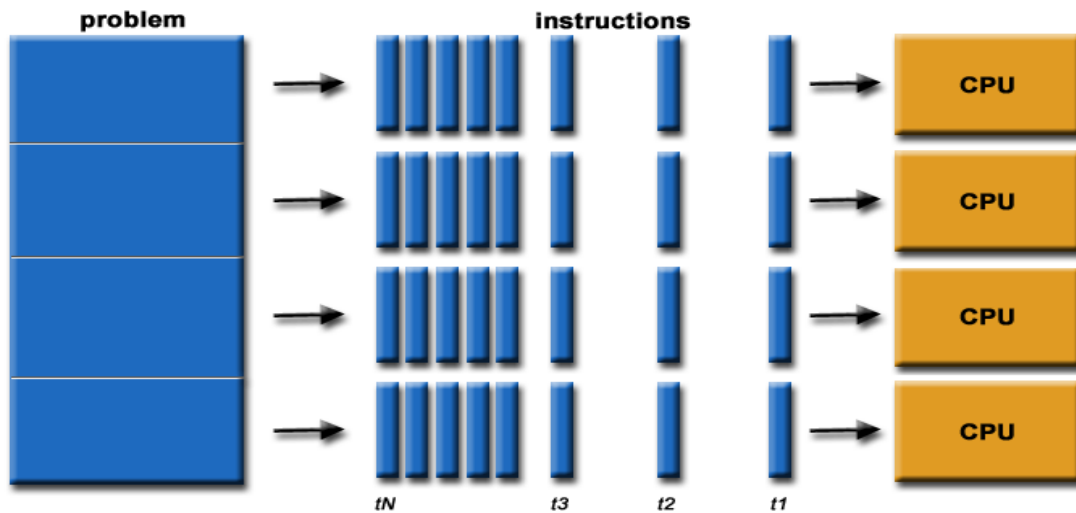


Figure 2.2

2.3 Previous Work

- i. Vector processors
- ii. Cloud Services by Amazon, Zoho
- iii. General-purpose computing on graphics processing units (GPGPU)
- iv. Reconfigurable computing with field-programmable gate arrays
- v. Automatic parallelization

2.4 Issues Solved by CPC

In MCS Cloud Computing has never been experimented on Unix based operating systems and there are sufficient Ultra Sparc Server Machine available to use them for providing cloud computing services.

Our System has provided Cloud Parallel Computing by configuring Java Parallel Processing Framework on Sun Ultra Sparc Server Machines as Master node and on Heterogeneous Slave nodes.

2.5 Roles and Responsibilities

Name	Role	Responsibility
Capt Naeem Amjad	Project Leader	Project Management, Providing resources, documentation and presentation, Providing Resources, Algorithm testing
NC Imtiaz Qumar	Team Member	Server and client end application development and Configuration, algorithms design
NC Sreman Qamar	Team Member	Server and client end application development and Configuration, algorithms design

Table 2.5

2.6 Work Breakdown Structure

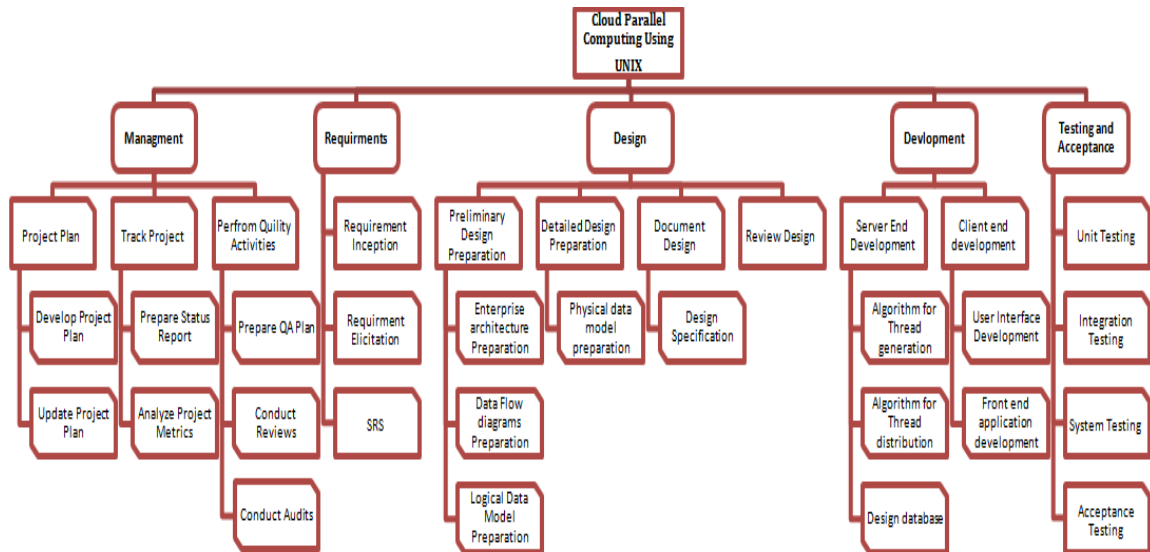


Figure 2.6

2.7 Project Plan

Task	Oct 2012	Nov 2012	Dec 2012	Jan 2013	Feb 2013	March 2013	April 2013	May 2013
Project Proposal								
Requirement Elicitation								
Software Requirement Specification								
Design document								
Establish a network of ultra workstations								
Server application for thread generation								
Server application for distribution								
Client side interface								
Testing and Acceptance								
Documentation								

Table 2.7

SYSTEM REUIREMENTS

3.1 Introduction

This chapter describes the functional and nonfunctional requirements of the Cloud Parallel Computing using Unix OS. Objectives are briefly summarized followed by detailed description of the system's scope, vision, use case, features and other related requirement issues.

3.2 Product Scope

This is a system to perform computation of a complex and time consuming task on separate machines i.e. by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously on separate machine.

There will be a master server that accepts a task from client, breaks it down into subtasks that can be executed in parallel and then executes these tasks on different machines then combining the results and giving the result back to client through an interface.

There will be a front end interface through which a client submits tasks and select resources. A middleware system at the backend will distribute the task work on multiple available machines.

3.3 Product Perspective

This will be a standalone system that runs on UNIX based Sun Ultra25 workstations. It will be responsible for receiving the jobs submitted by the user and delegating them to the appropriate workstations by splitting the job into parts that can be executed concurrently. CPC would not decide on its own how much workstation it will allow to one task, it's up to user to select the resources that suits his/her needs. After that CPC dispatches the job to the chosen workstation. Once a subtask has been assigned to a workstation, CPC is responsible for monitoring the progress and collecting the results from each workstation when the task is done and finally providing the results to end user. There will be no modifications to the UNIX Kernel.

3.4 Product Functions

The main purpose of the CPC is to perform computation of a complex and time consuming task submitted by user. The job details submitted by the user will include selection of number of workstations that the system should assign for one job. The system then allocates workstations selected by the user, enabling the CPC to minimize time needed to perform that job.

The overall functions of CPC are

- i. **Submit Job**
- ii. **View Job Status**
- iii. **Delete/Change Job**
- iv. **Initialize Job**
- v. **Determine Execution Host**
- vi. **Dispatch Job**
- vii. **Generate Report**

3.5 User Classes and Characteristics

There are essentially two classes of users for the CPC system: the user who wishes to submit jobs for the computing and the administrator who oversees scheduling and computing usage. The user needs to know the exact nature of the submitted job, such as the execution time as well as resources required, and must possess the technical knowledge about how to use the interface for submitting jobs. The administrator must be an advanced operator, fully qualified in using Solaris, and the CPC system.

3.6 Operating Environment.

Particulars	Client Side	Server Side
Operating System	Sun Solaris 10	Sun Solaris 10
Processor	1.34-GHz UltraSPARC IIIi processor	1.34-GHz UltraSPARC IIIi processor
RAM	1 GB	1 GB
Hard Drive	Customized	Customized

Table 3.6

3.7 Design and Implementation Constraints

Cloud Parallel Computing using Unix OS is based on Java Parallel Processing Framework which works on Unix/Window based system with above mentioned operating system..

Moreover it require JDK Version 1.5 or later and Apache Ant Version 1.7 or later. Network computers should support SSH and FTP.

3.8 User Documentation

A user manual is provided at the end of this document.

3.9 Assumption and Dependencies

External interface requirements specify hardware and software elements with which a system or component must interface

A number of factors that may affect the requirements specified in the SRS include:

- i. It is assumed that all workstations are running on Sun Solaris 10.
- ii. It is assumed that all users will have basic knowledge of Solaris or any other UNIX.
- iii. **Users** are assumed to have a fair estimate of job execution times, so that the decision to accept or reject a job is facilitated.

3.10 External Interface Requirements

3.10.1 User Interfaces

The minimal requirements are that the CPC user would be able to interact with the system through the command prompt, or through the graphical interface provided by the system. There will be a menu driven interface in command prompt mode for each of the following action.

- i. **S**ubmit jobs with the associated workstations.
- ii. **M**onitor the status of submitted jobs
- iii. **C**ancel/delete jobs submitted by him
- iv. **A**dd Node
- v. **D**elete Node/Restart Node

3.10.2 Hardware Interfaces

System involves only sun ultra25 workstations and it has no external hardware. The communication among the system involves only the networking infrastructure including

- i. Ethernet card
- ii. The switch

3.10.3 Software Interfaces

CPC will directly interact with Sun Solaris 10, and other IDE's and virtual servers for example

- i. Net Beans
- ii. Management and Monitoring Console
- iii. Java Parallel Programming Framework
- iv. Any Internet browser

3.10.4 Communication Interface

CPC will utilize the communications architecture of Sun Solaris itself, and will not have any unique communications interfaces.

The communication is performed over the network using Solaris communication utilities like Telnet and FTP

3.11 System Features

This section is organized by the processes and features encapsulated in CPC, following are the main use cases:

- i. Registration
- ii. Login
- iii. Submit Job
- iv. Job Initialization
- v. View Job Status
- vi. Delete Job
- vii. Delete Node
- viii. Add Node

- ix. Determine Execution Host
- x. Dispatch Job

3.11.1 Registration

3.11.1.1 Description and Priority

Users of the system need to register before using the cloud services. This functional feature deals with the end-user (can also be used by the administrator) and is facilitated by the interface of the CPC. The user details and other credentials are collected through this interface.

3.11.1.2 Stimulus/Response Sequences Actors: End users, administrator

Actor: End users, administrator

- i. User: The user accesses login the interface of system
- ii. System: The system asks the user to enter his username and password.
- iii. User: The user enters his username and password and presses submit button.
- iv. System: The system verifies the credentials from the database and either allows the user to use the cloud service or generates an error message in case of invalid details.

3.11.1.3 Functional Requirements

- i. The system should provide the user registration module along with an interface to access it.
- ii. The system should maintain a database of registered users.

3.11.2 Login

3.11.2.1 Description and Priority

Administrator and the other registered members require to login to the system before using the cloud services. After login they can do the further actions while using the cloud service

3.11.2.2 Stimulus/Response Sequences

Actors: Registered users, administrator

- i. User: The user accesses login the interface of system

- ii. System: The system asks the user to enter his username and password.
- iii. User: The user enters his username and password and presses submit button.
- iv. System: The system verifies the credentials from the database and either allows the user to use the cloud service or generates an error message in case of invalid details.

3.11.2.3 Functional Requirements

The system should provide an interface for the user to login and system user login module involves user authentication and session tracking

3.11.3 Submit Job

3.11.3.1 Description and Priority.

This functional feature deals with the registered user (can also be used by the administrator) who is login to the system and is facilitated by the interface of the CPC. The user input and other parameters required for the working of system are collected through this interface.

3.11.3.2 Stimulus/Response Sequences

Actors: Registered users, administrator

- i. User: The user accesses the job submission interface of system
- ii. System: The system asks the user to select the job and available resources.
- iii. User: The user provides the required input for the execution of the job including the resources.
- iv. System: The system acquires the user's input after validating it.

3.11.3.3 Functional Requirements

- i. The system should provide an interface for the user to submit a job.
- ii. The system should collect the job specification from the user

- iii. The system should maintain a database of submitted jobs.
- iv. The system should have a mechanism to validate the user inputs for job submission.

3.11.4 Job Initialization

3.11.4.1 Description and Priority.

System will split the job submitted by registered user or administrator to run in parallel on multiple machines of cloud..

3.11.4.2 Stimulus/Response Sequences

Actors: Registered users, administrator

- i. User: The user should have already submit the job
- ii. System: The system splits the defined job into chunks of code that can be executed in parallel on multiple machines of cloud

3.11.4.3 Functional Requirements

The System should have a module to split the job.

3.11.5 View Job Status

3.11.5.1 Description and Priority.

Registered users or administrator will be able to view the status of their submitted job.

3.11.5.2 Stimulus/Response Sequences

Actors: Registered users, administrator

- i. User: The user clicks the “view job status “button”.
- ii. System: The system will display the job status to user

3.11.5.3 Functional Requirements

- v. The system should provide an interface for the user to view job status.
- vi. The system should have a module asks the relevant machines about the status of job

- vii. The system should have a module that receive the status of job from relevant machines

3.11.6 Delete Job

3.11.6.1 Description and Priority.

Registered users or administrator will be able to delete their submitted job or stop it during its execution..

3.11.6.2 Stimulus/Response Sequences

Actors: Registered users, administrator

- i. **User:** The user clicks the “view job status “button”.
- ii. **System:** The system will display the job status to user.
- iii. **User:** The user then selects to stop/delete his job.
- iv. **System:** The system responds to the selected action after reconfirmation from user.

3.11.6.3 Functional Requirements

- i. The system should provide an interface for the user to view job status.
- ii. The system should have a module to send a message to relevant machines of the cloud to stop/delete the job.

3.11.7 Delete Node

3.11.7.1 Description and Priority.

Registered users or administrator will be able to delete the any node or stop it during its execution..

3.11.7.2 Stimulus/Response Sequences

Actors: Registered users, administrator

- v. **User:** The user clicks the “view job status “button”.
- vi. **System:** The system will display the job status to user.
- vii. **User:** The user then selects to stop/delete his Node.
- viii. **System:** The system responds to the selected action.

3.11.7.3 Functional Requirements

- iii. The system should provide an interface for the user to view job status.
- iv. The system should have a module to send a message to relevant machines of the cloud to stop executing.

3.11.8 Add Node

3.11.8.1 Description and Priority.

Registered users or administrator will be able to add Node Remotely to execute their Job

3.11.8.2 Stimulus/Response Sequences

Actors: Registered users, administrator

- ix. **User:** The user clicks the “view job status “button”.
- x. **System:** The system will display the job status to user.
- xi. **User:** The user then selects to add Node Remotely through SSH.
- xii. **System:** The system responds to the user action

3.11.8.3 Functional Requirements

- v. The system should provide an interface for the user to view job status.
- vi. The system should have a module to send a message to relevant machines of the cloud to add as a Node to CPC.

3.11.9 Dispatch Job

3.11.9.1 Description and Priority.

The system will delegate the initialized job to multiple machines of cloud by dispatching the each subtask to separate machines

3.11.9.2 Stimulus/Response Sequences

Actors: Master workstation

- i. **Master workstation:** The master workstation issues the dispatch command to multiple machines.
- ii. **System:** The will dispatch the sub tasks to relevant machines

3.11.9.3 *Functional Requirements*

The system should have a module to send a message to relevant machines of the cloud to dispatch the job

3.12 Other Nonfunctional Requirements

Nonfunctional requirements of the automatic report generating system consist of performance, security requirements.

3.12.1 Performance Requirements

The maximum response time for the submission of a job will be 2 minute.

3.12.2 Capacity

- i. The maximum number of jobs schedulable at a time is limited only workstations availability.
- ii. Safety Requirements
- iii. Load balancing must be ensured at every workstation in order to prevent the system from failure.
- iv. System should prevent any unauthorized access by any illegitimate user.
- v. System must not violate any law or the rights of any individual or entity, and must not expose product or users to harm or legal liability

3.12.3 Security Requirements

- i. Users will get login only by their own username & password.
- ii. System will ensure the privacy of user job status.
- iii. Only assigned administrator for specific user can view all jobs status.

3.13 Software Quality Attributes

Software quality attributes of automatic report generating system are robustness, usability measures and maintainability.

3.13.1 Reliability:

- i. There will be a maximum of 5 bug/KLOC.
- ii. In case of system crash, the CPC will be down while the Solaris operating system will run. This will take less than five minutes.
- iii. The system defect rate shall be less than 1 failure per 500 hours of operation

3.13.2 Usability:

- i. Four out of five users shall be able to submit job within 3 minutes after a 2-hour introduction to the system.
- ii. Novice users shall perform any task in 10 minutes. While experienced users shall perform any task in 2 minutes.
- iii. At least 80% of customers polled after a 3 months usage period shall rate their satisfaction with the system at 7 and more on a scale of 1 to 10.

3.13.3 Maintainability

- i. No method in any object may exceed 300 lines of code.
- ii. Installation of a new version shall leave all database contents and all personal settings unchanged

3.13.4 Testability:

- i. The delivered system shall include unit tests that ensure 98% branch coverage.
- ii. Development must use regression tests (seeks to uncover new errors) allowing for full retesting in 5 hours.

3.13.5 Availability:

- i. The system shall meet or exceed 97.99% uptime.

- ii. The system shall not be unavailable more than 1 hour per 500 hours of operation.
- iii. Less than 40 seconds shall be needed to restart the system after a failure 95% of the time.

3.14 Other Requirements

All requirements are already mentioned above.

System Design

4.1 Introduction

System Design chapter provides a comprehensive architectural overview of the Cloud Parallel Computing Using Unix OS. It presents various architectural views to show different aspects of the system. It is intended to capture and communicate the significant architectural decisions which have been made on the system.

This chapter will display the logical, implementation and dynamic view with use cases of the system in order to show the complete system.

4.2 Scope

The scope of this document is to depict the architecture of the Cloud Parallel Computing Using Unix OS. This will allow various stakeholders to find what they need in the software architecture.

4.3 Architecture Design

4.3.1 Architecture Pattern:

This system uses distributed memory model to perform network parallel Computing. The communication among systems is done through Message Passing Interface (MPI).

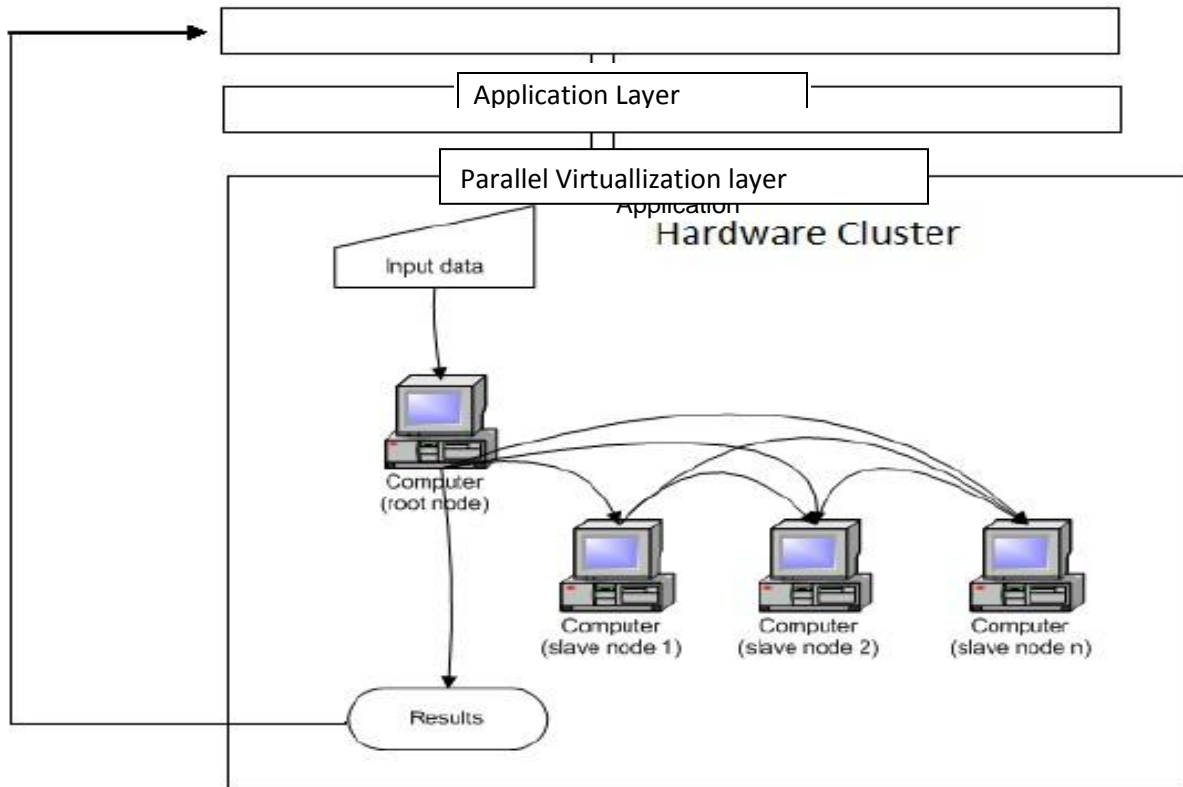


Figure 4.3.1

Architecture Pattern

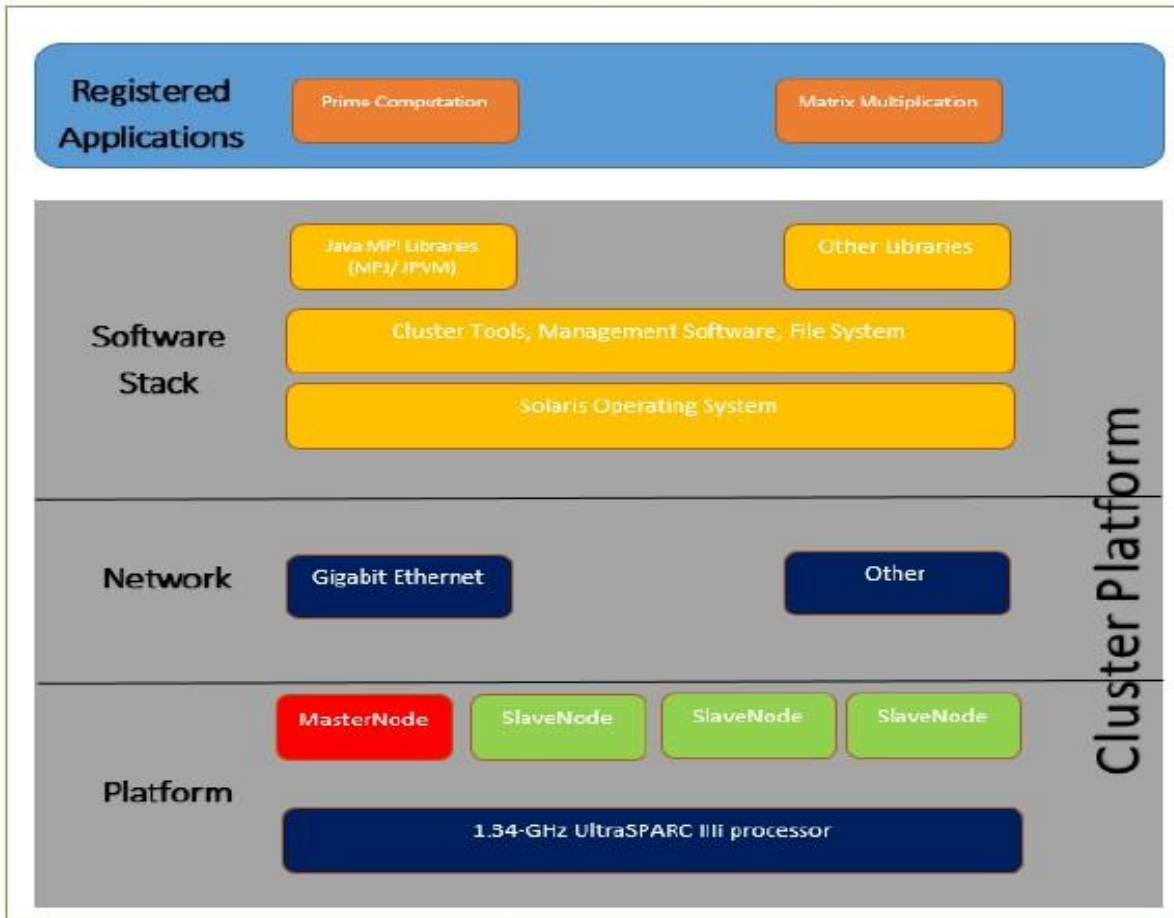


Figure 4.3.2

4.4 Detailed Design

Detail design of Cloud Parallel Computing system consists of uses cases, class diagram, sequence diagram and activity diagram.

4.4.1 ER Diagram

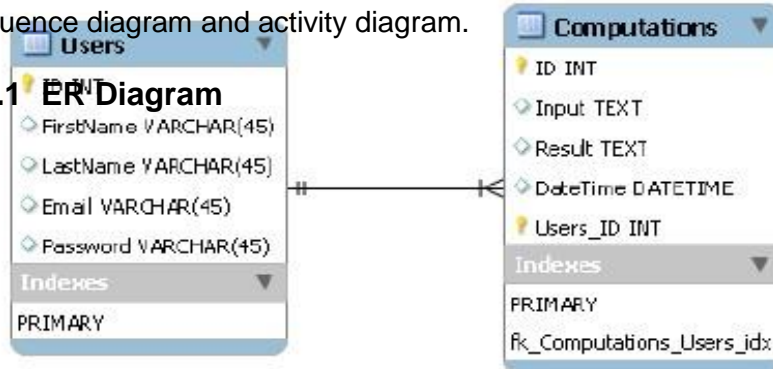


Figure 4.4.1

The above entity relationship diagrams represent two entities in our system, The users and the computations. The Users Table contains the information of registered users and the computations table contains the history of the computations performed by a registered user

4.4.2 Class Diagram

The Main classes of our system are

- i. MasterNode
- ii. SlaveNode
- iii. Message
- iv. Task
- v. Environment
- vi. The communication among nodes is done through parallel virtual machines libraries

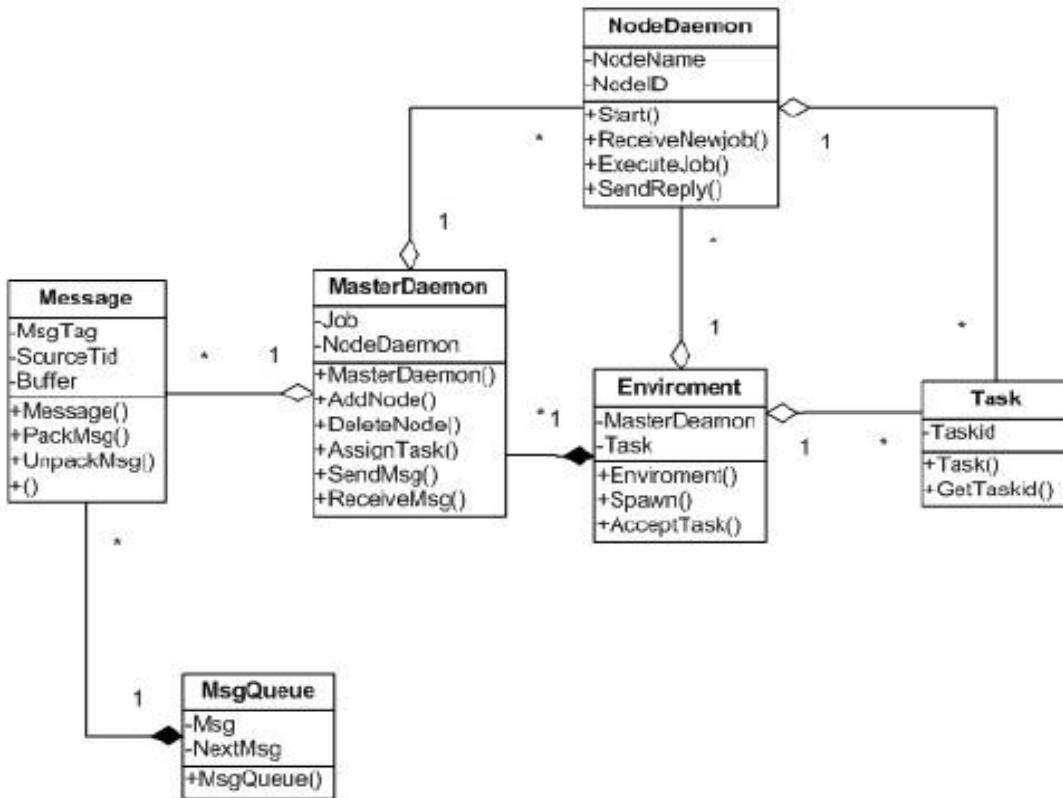


Figure 4.4.2

4.4.3 Use Cases:

Use cases show the interaction between the actors and system to achieve a goal. Use cases of Cloud Parallel Computing are shown in figure 4.2 and figure 4.3. Actors and use cases of CPC are.

Actors: User, Admin

Use Cases: Login, Adm Login, Logout, , Submit Job, Distribute Tasks, Receiver Results, Execute Tasks.

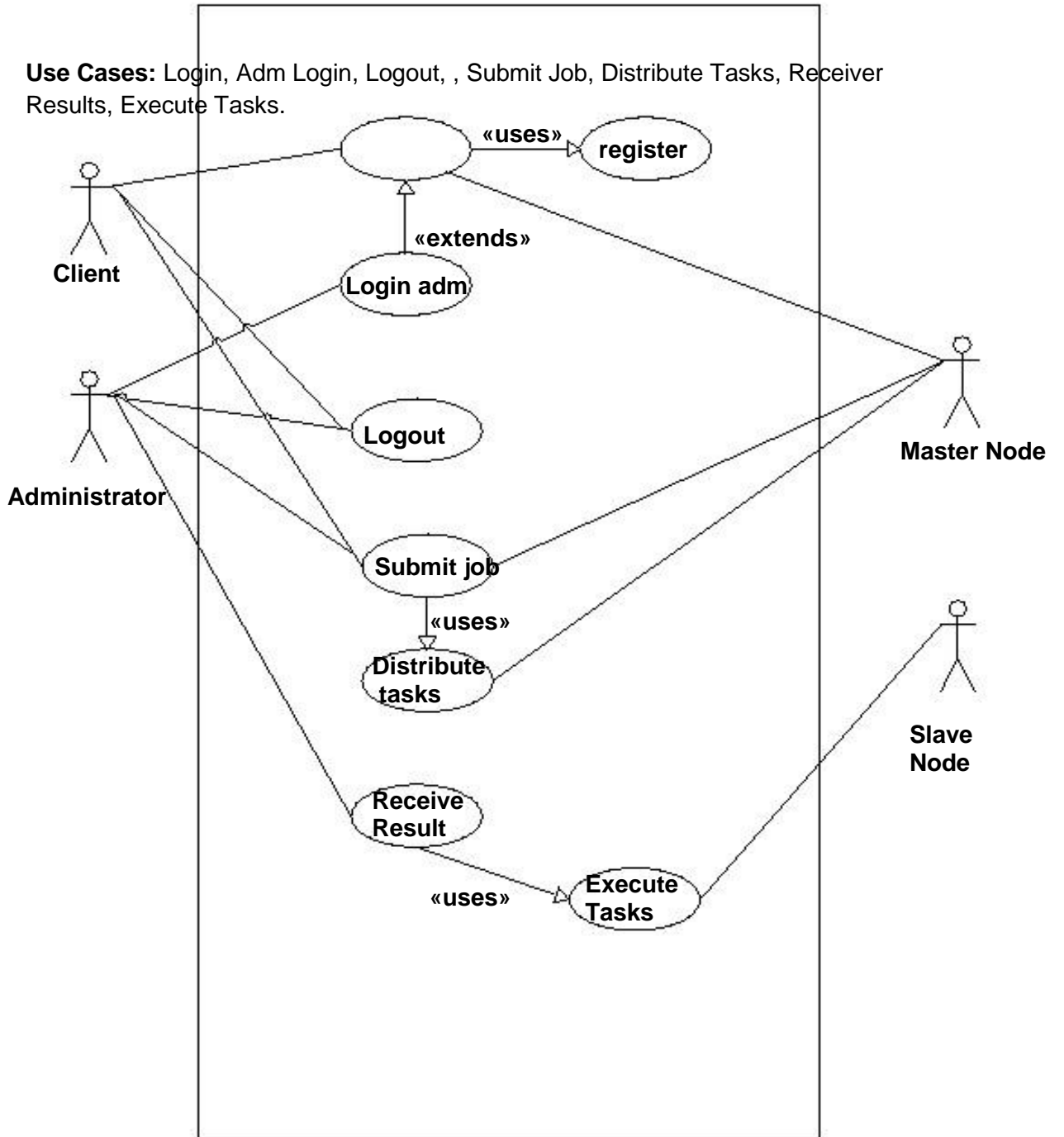


Figure 4.4.3: Administrator /Client Use case

4.4.3.1 Use Case Descriptions:

Use case descriptions provide the details of each use case.

i. Login:

Actor (Initiator): Client.

Purpose: To select to login to system.

Overview: System will open shell session for login

Preconditions: System must be running.

Flow of Events:

Actor Action: Enter valid user name and password.

System Response: System will login the client.

Post Conditions: System is running.

ii. Adm Login:

Actor (Initiator): Administrator.

Purpose: To Login the adm.

Overview: System will provide with a remote shell to login.

Preconditions: Selected Shell to open.

Flow of Events:

Actor Action: Enter credentials.

System Response: Login the adm.

Post Conditions: System is running.

iii. Submit Job:

Actor (Initiator): Administrator, Client.

Purpose: To submit a job.

Overview: System will provide with a remote access to server to submit a job.

Preconditions: Selected Shell to open.

Flow of Events:

Actor Action: Enter credentials and submit a job.

System Response: Submit the job.

Post Conditions: System is running.

iv. **Logout:**

Actor (Initiator): Administrator, Client.

Purpose: To Logout the user.

Overview: System will provide with a remote shell to logout.

Preconditions: Selected Shell to open.

Flow of Events:

Actor Action: Enter credentials.

System Response: Login the user.

Post Conditions: User Logged in.

v. **Register:**

Actor (Initiator): Administrator, Client.

Purpose: To Register the user.

Overview: System will provide with a remote shell to Register.

Preconditions: Selected Shell to open.

Flow of Events:

Actor Action: Enter credentials.

System Response: Register the user.

Post Conditions: System is running.

vi. **Distribute Task:**

Actor (Initiator): Administrator, Client.

Purpose: To distribute a job into tasks.

Overview: System will provide with a remote access to server to submit a job.

Preconditions: Selected Shell to open.

Flow of Events:

Actor Action: Enter submit a job.

System Response: Submit the job and distribute it into task to different nodes..

Post Conditions: System is running.

vii. **Distribute Task:**

Actor (Initiator): Administrator, Client.

Purpose: To execute tasks.

Overview: System will provide with a remote access to server to submit a job.

Preconditions: Selected Shell is open and job submitted.

Flow of Events:

Actor Action: Enter submit a job.

System Response: Submit the job and distribute it into task to different nodes to execute it.

Post Conditions: System is running and job is submitted

viii. **Receive Results:**

Actor (Initiator): Administrator, Client.

Purpose: To receive result of a computation.

Overview: System will provide with a remote access to server to submit a job.

Preconditions: Selected Shell to open and job submitted.

Flow of Events:

Actor Action: Enter submit a job.

System Response: Submit the job and distribute it into task to different nodes and receive results.

Post Conditions: System is running and job submitted.

4.4.4 Sequence Diagram

Sequence diagrams shows how process operates with each other and operates in which order

4.4.4.1 Login Sequence Diagram

Login Sequence diagram of Cloud Parallel Computing is shown at figure 4.5.

- i. Main page GUI redirect user/client to login page.
- ii. Here user/client will input his/her information.
- iii. Information entered by user/client goes to Master Node.
- iv. Master Node will send that information to Database for validation.

- v. If information entered is correct database will acknowledge with true Boolean value else false.

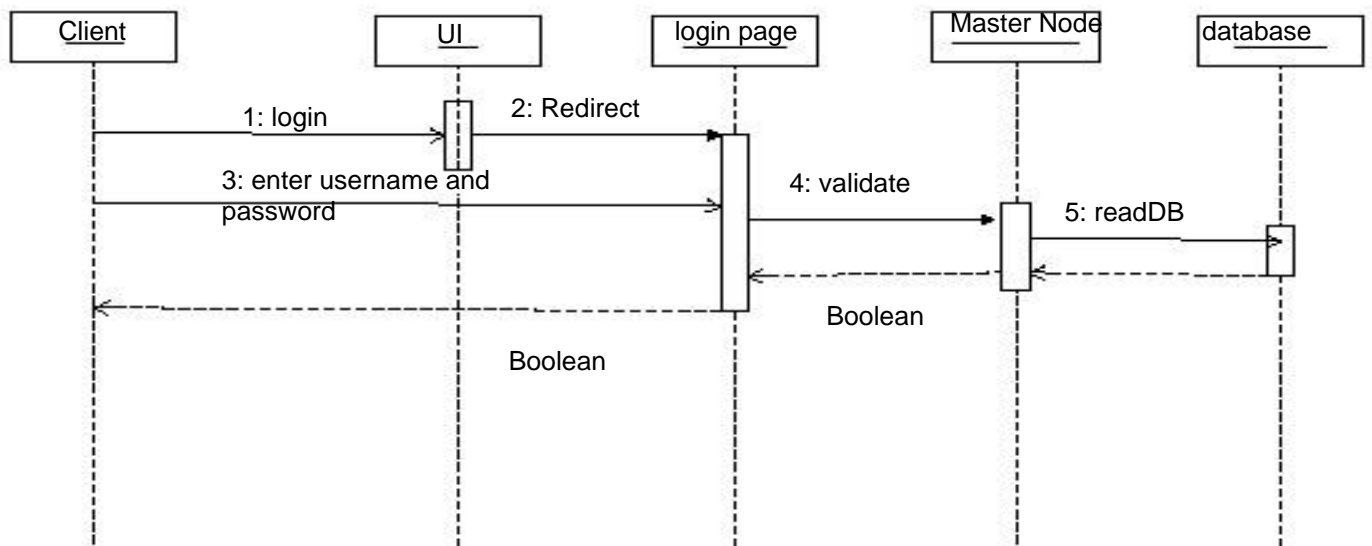


Figure 4.4.4.1:Sequence Diagram

4.4.4.2 System Sequence Diagram

Login Sequence diagram of Cloud Parallel Computing is shown at figure 4.6.

Flow of the system is as follow;

- i. Registered user commands Main GUI for job submission.
- ii. Main GUI will redirect user to Job submit page and he/she will enter valid data..
- iii. If user is already on Job submit page he/she will enter valid data.
- iv. Job submit page will forward that job to MasterNode.
- v. Master Node will split job into tasks.
- vi. MasterNode then assign tasks to SlaveNodes.
- vii. SlaveNodes will execute the tasks and send it to the MasterNode.
- viii. MasterNode will compile the result and send it to the Job submit page.
- ix. Job submit will show result to user.

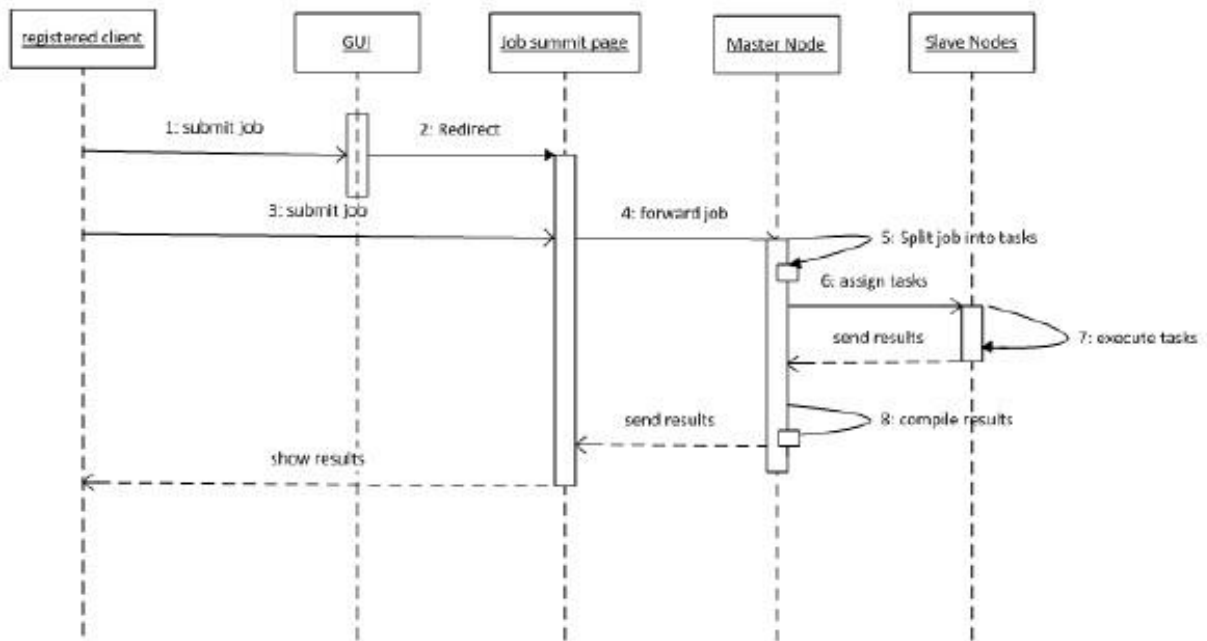


Figure 4.4.4.2

4.4.5 Activity Diagram:

Activity diagram shows the graphically workflows of stepwise activities with options. Activity diagram of automatic report generating system is shown at figure 4.6.

Description of Sign-up:

- i. Precondition: User is not registered to the system.
- ii. Client Entered Username: If entered username is available
- iii. Invalid Username: If entered username is not valid or already taken user will be redirected to the
- iv. Signup page.
- v. Entered Password and Retype: Client will enter password and retype it if both passwords are same

- vi. System will not generate error else system will generate error.
- vii. Client logged in: Client is successfully logged in the system.

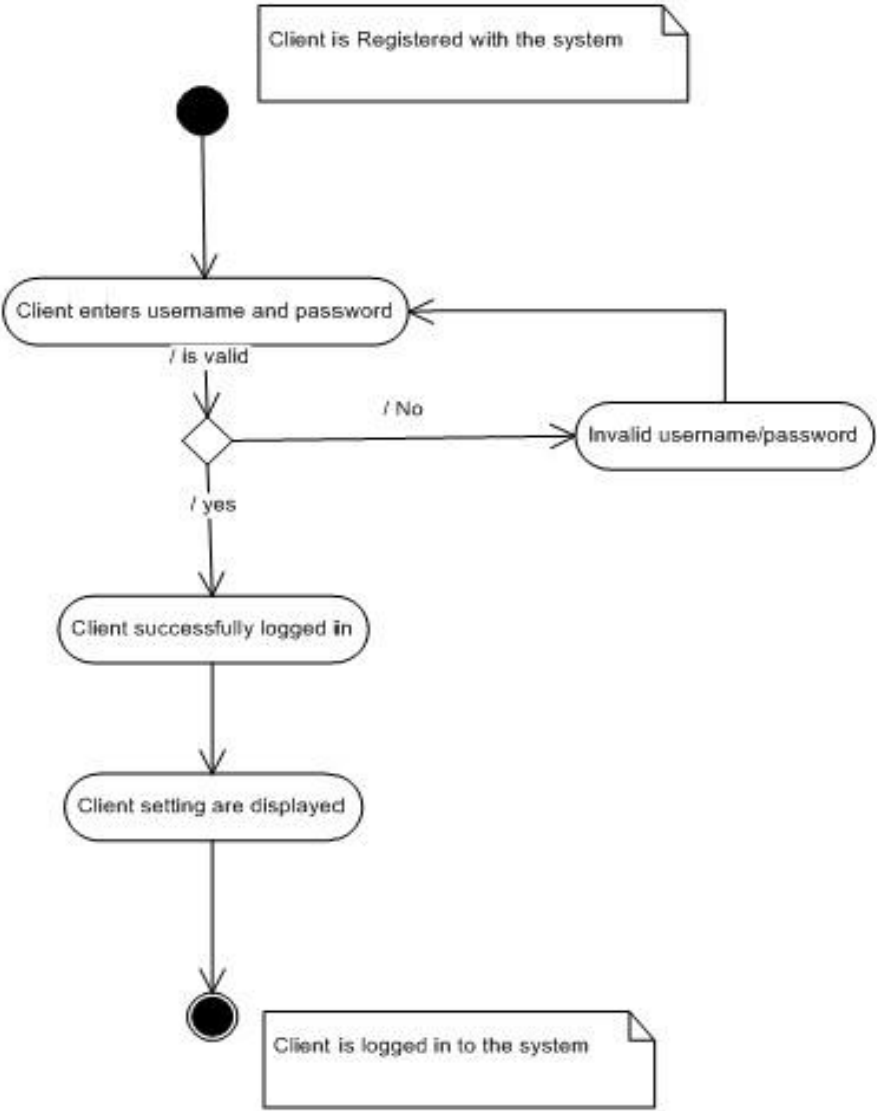


Figure 4.4.5:Activity Diagram

SYSTEM IMPLEMENTATION

5.1 Tools and Technologies

5.1.1 Apache ANT

Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications. Ant can also be used effectively to build non Java applications, for instance C or C++ applications.

5.1.2 Java Development Kit

The Java Development Kit (JDK) is an implementation of either one of the Java SE, Java EE or Java ME platforms

5.1.3 PuTTY

PuTTY is a free implementation of Telnet and SSH for Windows and Unix platforms

5.1.4 Shell Scripting

A shell script is a script written for the shell, or command line interpreter, of an operating system. The shell is often considered a simple domain-specific programming language. Typical operations performed by shell scripts include file manipulation, program execution, and printing text.

5.1.5 WinScp

Open source graphical SFTP client for Windows

5.1.6 Secure Shell (SSH)

Secure Shell (SSH) is a cryptographic network protocol for secure data communication, remote shell services or command execution and other secure network services between two networked computers that connects, via a secure channel over an insecure network, a server and a client (running SSH server and SSH client programs, respectively). The protocol specification distinguishes between two major versions that are referred to as SSH-1 and SSH-2.

5.2 Software Implementation

The system is a distributed application implemented in java.

The System has four main modules. The detail of each module is given below.

5.2.1 Administration and Monitoring Console

Administration Console is a java application designed to monitor the job status and execution time and results.

5.2.2 Server Distribution

Cloud Parallel Computing Server Application is a java Application with a command Line Interface.

This application start the Cloud Server

The main functionalities of a cloud server are

- i. Initializing the Parallel programming environment
- ii. Accept Connection from a new node and add it to the existing environment
- iii. Accept Connection request from a client and receive program data from client
- iv. Prioritize the Connections and the job responsibilities
- v. Maintain thread pool containing tasks of a job.
- vi. Forward each thread data and code to an available node machine in Cloud Environment
- vii. Accept results from each worker or node machine
- viii. Combine the results of a job by collecting the result from individual node machine
- ix. Send the job results back to the client

5.2.3 Node Distribution

Cloud Parallel Computing Node Application is a java based Application with Command Line Interface

This Application offer the resources of a machine to cloud environment

Main features of Node application are as given below

- I. Activate the Node Machine
- II. Request a connection to the Server of Parallel Environment
- III. Waiting for task assignment from server Machine
- IV. Accepts Task assignment from Server machine
- V. Execute the task received from a server locally using Java Development Kit
- VI. Return the results back to the server

5.2.4 Client Application

This is a java based application developed to test different features offered by cloud Parallel System. This application has a command line interface

Main features of this application are given below

- i. This application is programmed to split is a single job of matrix multiplication into a number of tasks
- ii. It maintain a thread pool containing tasks of a single processor hungry job.
- iii. It requests a connection to the Parallel Cloud Server and submits the job to the server

5.3 Implementation Terminologies and Code Snippets

5.3.1 Task Objects

A task is the smallest unit of execution that can be handled by the framework. We will say that it is an atomic execution unit. A JPPF application creates tasks, groups them into a job, and submits the job for execution on the cloud

5.3.2 JPPFTask

JPPFTask is the base super class for any task that is run by JPPF.

JPPFTask is defined as follows:

```
public abstract class JPPFTask implements Task<Object> {  
  
    ...  
  
}
```

We can see that this class implements the Task<T> interface, defined as follows:

```
public interface Task<T> extends Runnable, Serializable {  
  
    ...  
  
}
```

We have outlined three important keywords that characterize JPPFTask:

- **abstract**: JPPFTask cannot be used directly, it must be extended to construct a real task
- **Runnable**, via Task<Object>: when writing a JPPF task, the run() method of java.lang.Runnable must be implemented. This is the part of a task that will be executed on a remote node.
- **Serializable**, via Task<Object>: tasks are sent to servers and nodes over a network. JPPF uses the Java serialization mechanism to transform task objects into a form appropriate for networking

To write a real task in your application, you simply extend JPPFTask to implement your own type:

```
public class MyTask extends JPPFTask {  
  
    public void run() {  
  
        // ... code here ...  
  
    }  
  
}
```

5.3.3 Job name and Identifier

Each job has a unique identifier (UUID) that allows JPPF to manage and monitor the job while distinguishing it from other jobs. If this identifier is not explicitly specified via a dedicated constructor, JPPF will create one as a string of 32 hexadecimal characters. It is very important that all jobs across an entire JPPF grid have a unique distinct uuid, otherwise there is no guarantee that a job will be executed properly.

Additionally, a job can have a name which doesn't need to be unique, and which is used by the JPPF administration console for display purposes only. You may also use it in your application for logging and tracing. If not set by the user, the name will be by default equal to the uuid.

The class `JPPFJob` provides the following APIs for the job name and uuid:

```
public class JPPFJob implements Serializable, JPPFDistributedJob {  
  
// create a blocking job with the specified uuid  
  
public JPPFJob(final String jobUuid)  
  
// get this job's UUID  
  
public String getUuid()  
  
  
  
// get the user-defined display name for this job  
  
public String getName()  
  
// set the user-defined display name for this job  
  
public void setName(final String name)  
  
}
```

5.3.4. Creating a job

To create a job, the JPPFJob class offers a number of constructors, which can be split in 2 groups:

Constructors for blocking jobs

```
// creates a blocking job with no data provider and default SLA values
public JPPFJob()

// creates a blocking job with the specified data provider and default SLA values
public JPPFJob(DataProvider dataProvider)

// creates a blocking job with the specified data provider and SLA
public JPPFJob(DataProvider dataProvider, JPPFJobSLA jobSLA)
```

Constructors for non-blocking jobs

```
// creates a blocking job with the specified execution results listener,
// no data provider and default SLA values
public JPPFJob(TaskResultListener resultsListener)

// creates a blocking job with the specified execution results listener,
// data provider and default SLA values
public JPPFJob(DataProvider dataProvider, TaskResultListener resultsListener)

// creates a blocking job with the specified execution results listener,
// data provider and SLA
public JPPFJob(DataProvider dataProvider, JPPFJobSLA jobSLA,
              TaskResultListener resultsListener)
```

Basically, the distinction for a non-blocking job is made via the presence of a TaskResultListener.

Finally, there is a more generic constructor that embraces everything the other constructors do:

```
// creates a job with the specified data provider, SLA, blocking
indicator
// and execution results listener
public JPPFJob(DataProvider dataProvider, JPPFJobSLA jobSLA, boolean blocking,
              TaskResultListener resultsListener)
```


No matter which constructor is used, the job id is automatically generated as a pseudo-random string of 32 hexadecimal characters. It can then be obtained or changed with the job's `getId()` and `setId(String)` methods. This mechanism ensures that a job always has an id, and that developers always have the possibility to change it to a more readable one.

5.3.5. Adding tasks to a job

```
public JPPFTask addTask(Object taskObject, Object...args) throws JPPFException
```

The `taskObject` parameter can be one of the following:

- i. an instance of `JPPFTask`
- ii. an instance of a class with a non-static public method annotated with `@JPPFRunnable`
- iii. a `Class` object representing a class that has a public static method or a constructor annotated with `@JPPFRunnable`
- iv. an instance of a `Runnable` class
- v. an instance of a `Callable` class

The `args` parameter is optional and is only used to pass the arguments of a method or constructor annotated with `@JPPFRunnable`. It is ignored for all other forms of tasks.

The return value is an instance of (a subclass of) `JPPFTask`, regardless the type of task that is added.

As JPPF is using reflection to properly wrap the task, an eventual exception may be thrown. It will then be wrapped into a `JPPFException`.

5.3.6. Handling of execution results

The results of the tasks executions are handed over by an instance of `TaskResultListener`.

We can thus say the `TaskResultListener` is an asynchronous receiver, whose job is to:

- i. handle and store the execution results of the tasks
- ii. ensure the results are in the same order as the tasks initially submitted
- iii. handle errors occurring while receiving the results from the server
- iv. update the state of the job's execution
- v. optionally handle the persistence of the job's state for later recovery

To store the execution results, the JPPFJob class holds an instance of JobResults, which is accessible via the getResults() method. JobResults provides the following API:

```
public class JobResults {
// Get the current number of received results
    public synchronized int size()

// Determine whether the job received a result
// for the task at the specified position
    public synchronized boolean hasResult(final int position)

// Add the specified results to the job
    public synchronized void putResults(final List<JPPFTask> tasks)

// Get all the tasks received as results for the job
    public synchronized Collection<JPPFTask> getAll()
}
```

Since JPPFResultCollector holds a reference to the job, it will be able to update the execution results each time it receives a resultsReceived(TaskResultEvent) notification.

A JPPF client is an object that will handle the communication between the application and the server. Its role is to:

- i. manage one or multiple connections with the server
- ii. submit jobs and get their results
- iii. handle notifications of job results
- iv. manage each connection's life cycle events
- v. provide the low-level machinery on the client side for the distributed class loading mechanism
- vi. provide an access point for the management and monitoring of each server

A JPPF client is represented by the class JPPFClient. We will detail its functionalities in the next sub-sections.

5.3.7. Creating and closing a JPPFClient

A JPPF client is a Java object, and is created via one of the constructors of the class JPPFClient. Each JPPF client has a unique identifier that is always transported along with any job that is submitted by this client. This identifier is what allows JPPF to know from where the classes used in the tasks should be loaded. In effect, each node in the grid will have a map of each client identifier with a unique class loader, creating the class loader when needed. The implication is that, if a new client identifier is specified, the

classes used in any job / task submitted by this client will be dynamically reloaded. This is what enables the immediate dynamic redeployment of code changes in the application. On the other hand, if a previously existing identifier is reused, then no dynamic redeployment occurs, and code changes will be ignored (i.e. the classes already loaded by the node will be reused), even if the application is restarted between 2 job submissions.

There are two forms of constructors for JPPFClient, each with a specific corresponding semantics:

Generic constructor with automatic identifier generation

```
public JPPFClient()
```

When using this constructor, JPPF will automatically create a universal unique identifier (uuid) that is guaranteed to be unique on the grid. The first submission of a job will cause the classes it uses to be dynamically loaded by any node that executes the job.

Constructor specifying a user-defined client identifier

```
public JPPFClient(String uuid)
```

In this case, the classes used by a job will be loaded only the first time they are used, including if the application has been restarted in the meantime, or if the JPPF client is created from a separate application. This behavior is more adapted to an application deployed in production, where the client identifier would only change when a new version of the application is deployed on the grid. It is a good practice to include a version number in the identifier.

As a JPPFClient uses a number of system and network resources, it is recommended to use it as a singleton. It is designed for concurrent use by multiple threads, which makes it safe for use with a singleton pattern. It is also recommended to release these resources when they are no longer needed, via a call to the JPPFClient.close() method. The following code sample illustrates what is considered a best practice for using a JPPFClient:

```
public class MyApplication {  
// singleton instance of the JPPF client
```

```

private static JPPFClient jppfClient = new JPPFClient();

// allows access to the client from any other class
public static JPPFClient getJPPFClient() {
    return jppfClient;
}

public static void main(String...args) {
// enclosed in a try / catch to ensure resources are properly released
    try {
jppfClient = new JPPFClient();

// ... application-sepcific code here ...
    } finally {
// close the client to release its resources
        if (jppfClient != null) jppfClient.close();
    }
}
}

```

5.3.8. Submitting a job

To submit a job, JPPFClient provides a single method:

```
public List<JPPFTasks> submit(JPPFJob job)
```

This method has two different behaviors, depending on whether the job is blocking or non-blocking:

- i. Blocking job: the submit() method blocks until the job execution is complete. The return value is a list of tasks with their results, in the same order as the tasks that were added to the job.
- ii. Non-blocking job: submit() returns immediately with a null value. It is up to the developer to collect the execution results by the means of a TaskResultListener set onto the job (see section Non-blocking jobs).

5.3.9. Cancelling a job

The ability to cancel a job is provided by JPPFClient's superclass AbstractGenericClient, which provides a cancelJob() method, defined as follows:

```

// superclass of JPPFClient
public abstract class AbstractGenericClient extends AbstractJPPFClient {
// cancel the job with the specified UUID
    public boolean cancelJob(final String jobUuid) throws Exception;
}

```

This method will work even if the client is connected to multiple drivers. In this case, it will send the cancel request to all the drivers.

5.3.10. Receiving notifications for new and failed connections

The JPPF client emits an event each time a new connection is established with a server. It is possible to receive these events by registering an implementation of the listener interface `ClientListener` with the client. Since the connections are generally established during the initialization of the client, i.e. when calling its constructor, `JPPFClient` provides a different form of the two constructors we have seen in `Creating and closing a JPPFClient` :

```
// Initialize with the specified listeners and a generated uuid
public JPPFClient(ClientListener...listeners)
// Initialize with the specified listeners and user-defined uuid
public JPPFClient(String uuid, ClientListener...clientListeners)
```

It is also possible to add and remove listeners using these two more "conventional" methods:

```
// register a listener with this client
public void addClientListener(ClientListener listener)
// remove a listener from the registered listeners
public synchronized void removeClientListener(ClientListener listener)
```

Here is a sample `ClientListener` implementation:

```
public class MyClientListener implements ClientListener {
    @Override
    public void newConnection(ClientEvent event) {
// the new connection is the source of the event
        JPPFClientConnection connection = event.getConnection();
        System.out.println("New connection with name " + connection.getName());
    }

    @Override
    public void connectionFailed(ClientEvent event) {
        JPPFClientConnection connection = event.getConnection();
        System.out.println("Connection " + connection.getName() + " has failed");
    }
}

ClientListener myClientListener = new MyClientListener();
// initialize the client and register the listener
JPPFClient jppfClient = new JPPFClient(myClientListener);
```

TESTING AND RESULT ANALYSIS

This Chapter shows the test cases designed to test the overall functionality of the system.

6.1 Testing

Testing not only maintains the software and system quality but also improves over all usability and stability of the project. At different stages of development suitable testing techniques were used to ensure product worked accurately and efficiently. Almost all the errors detected during testing were removed.

The overall approach of this test plan is Gray Box testing i.e. hybrid of Black Box and White Box testing. Testing approach will be same for all features of the system.

Test case no.1

Name: Login to Server.

Description: To test the login functionality to Server or Establish SSH Connection to Server.

Precondition: System shall be correctly installed and working.

Steps:

- i. For Windows user enters values required by Putty.
- ii. For UNIX user go in to “bash” mode write SSH command.
- iii. System will need password to login in to server.

Expected Output: Output should be according to the one expected by the user, user shall logged into the system.

Result: Pass

Test case no.2

Name: Run Server

Description: It will test is Server is running.

Precondition: SSH connection to should be established.

Steps:

- i. Go to directory where JPPF server is.
- ii. Start server by following commands.

startDriver.bat (for windows)

startDriver.sh (for UNIX)

ant (used in both OS)

Expected Output: Server should start running.

Result: Pass

Test case no.3

Name: Add Node

Description: It will test is node added alright.

Precondition: System shall be correctly installed and working and Server should be up.

Steps:

- i. Go to Directory where JPPF node is.
- ii. Start node by following commands:

startNode.bat (for Windows)

startNode.sh (for UNIX)

ant (works with both)

Expected Output: Node should be added to server.

Result: Pass

Test case no.4

Name: Create FTP Session with Server

Description: It will test the FTP session is created and working alright.

Precondition: Server shall be correctly installed and working.

Steps:

- i. For Windows User opens WinSCP.exe.
- ii. User fills all fields required by WinSCP.
- iii. For UNIX user open terminal, go to “bash” mode and write commands needed to establish FTP session.

Expected Output: FTP session should be created with server and user can move files.

Result: Pass

Test case no.5

Name: Run Matrix Multiplication Application.

Description: It will test is matrix multiplication running properly.

Precondition: Server should be running and at least one node is add to server.

Steps:

- i. Go to directory where application is.
- ii. Start application by following commands:

run.bat (for Windows)

run.sh (for Unix)

ant (works with both)

Expected Output: System should provide desired output of application.

Result: Pass

Test case no.6

Name: Shut Down Node

Description: It will test shutdown functionality of node.

Precondition: Server should be up and at least one node is added.

Steps:

- i. User opens Admin UI.
- ii. User selects tab “topology”.
- iii. Then shutdowns particular node.

Expected Output: Node should be shutdown (stop working as part of cloud).

Result: Pass

Test case no.7

Name: Restart Node

Description: It will test restart functionality of node.

Precondition: Server should be up and at least one node is added.

Steps:

- i. User opens Admin UI.
- ii. User selects tab "topology".
- iii. Then restart particular node.
 - a. Expected Output: Node should be restarted.

Result: Pass

6.2 Results and Analysis

The Cloud Parallel Computing Using Unix OS has been developed to work most efficiently in the Military College of Signals. It is supposed to save time of heavy Mathematical Calculation and serving as a platform for application developer to develop their own application using JPPF. It has to provide its users with the freedom to add definitions of their own functions, and running their own application on a well configured platform. This System gave excellent result once tested for an application of matrix multiplication which involves hundreds of calculations.

6.2.1 Results

The System is tested using variable number of nodes and the performance of system by running a simple 1000*1000 Matrix Multiplication Application are given below.

The execution time in seconds approximately

1 node: 31 to 33 seconds

2 nodes: 20 to 24 seconds

3 nodes: 11 to 16 seconds

6.2.2 Analysis

Since the system is based on Java which is an open source application development tool and a simple language generation performance, reliability and usability are important features. Performance is an important aspect of this system thus it is its special operation. The system is kept simple and its use very easy. The system consists of simple GUI to provide user with the ability to monitor the performance of all nodes involved in executing the job. The experiment shows how much faster a large job can be perform by adding more nodes to the task. Although communication time and network delay plays an important part in total execution time.

CONCLUSION AND FUTURE WORK

The goal of the project “Cloud Parallel Computing Using Unix OS” was to learn about Unix and develop and configure a Cloud for running the applications parallel thus reducing the overall execution time using Suns Ultra Sparc Workstations as Server Node and Network Computers based on Heterogeneous OS as Slave Node. This project was select to provide MCS students with an application platform which will serve as a basis for them to further develop and run their own parallel applications.. We envisioned developing a system which is dynamic and scalable on demand and which can be up and running in minutes. Sun Ultra Sparc Workstation of MCS CSE Lab which were not been utilize uptil now are now fully operational and been used as JPPF server. We received the goal at the end but faced certain issue related to the resources and lack of help specific to the domain of cloud parallel computing. The limitations forced us to limit our system to minimum possible workstation. But nevertheless our efforts have paid us and we got a system in working which was never been developed in MCS.

A flexible and open source platform is well configured and ready to be used by any application developers. Our team intends to give full help and support to this project and any other team which is intended to further work on our project. We have our all hopes that one day Pakistani nation will be among the advanced nations in the field of information technology, research and science and Pakistan will gain its respect amongst all modern nations of the world. Aamin!

References:

[1] Cloud Computing Explained: Implementation Handbook for Enterprises (John Rhoton)

[2] Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide (David S. Linthicum)

[3] Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide (David S. Linthicum)

[4] Parallel Computing: Principles and Practice

[5] Parallel Computing: Principles and Practice

[5] <https://computing.llnl.gov/tutorials/mpi/>

[6] <https://computing.llnl.gov/tutorials/mpi/>

[7] <http://www.jppf.org/doc/v2/index.php?title=Introduction>

[8] http://en.wikipedia.org/wiki/Message_passing

[9] <https://computing.llnl.gov/tutorials/mpi/>

Appendix A: USER MANUAL

TABLE OF CONTENTS

	<u>Page #</u>
1 System Overview	1-1
2 How To Start Server	2-1
2.1 Creat SSH Session With Server Via Putty.....	2-1
2.2 Run Server.....	2-2
2.3 Creat FTP Session With Server Via WinSCP	2-2
3 Adding Node.....	3-1
4 Run Application.....	4-1
5 Admin UI	5-1
5.1 Starting Admin UI.....	5-1
5.2 Shut Down or Restart Node	5-2
5.3 Set Thread Pool Size and Priority For Node.....	5-3
5.5 Charts Configuration	5-4
6 How to Install Ant.....	6-1

System Overview

1.0 System Overview:

The main purpose of the Cloud Parallel Computing is to perform computation of a complex and time consuming task submitted by user. The job details submitted by the user will include selection of number of workstations that the system should assign for one job. The system then allocates workstations selected by the user, enabling the Cloud Parallel Computing (CPC) to minimize time needed to perform that job.

The overall functions of CPC are

- i. **Submit Job**
- ii. **View Job Status**
- iii. **Delete/Change Job**
- iv. **Initialize Job**
- v. **Determine Execution Host**
- vi. **Dispatch Job**
- vii. **Add Node**
- viii. **Restart/Shut Down Node**

PreRequisites:

JPPF works on any system that supports Java. There is no operating system requirement, it can be installed on all flavors of Unix, Linux, Windows, Mac OS, and other systems such as OS or other mainframe systems.

JPPF requires the following installed on your machine:

- Java Standard Edition version 1.6 or later, with the environment variable JAVA_HOME pointing to your Java installation root folder
- Apache Ant, version 1.7.0 or later, with the environment variable ANT_HOME pointing to the Ant installation root folder
- Entries in the default system PATH for JAVA_HOME/bin and ANT_HOME/bin

Installation:

The JPPF distribution includes a number of standalone modules or components, which can be deployed and run independently from any other on separate machines, and/or from a separate location on each machine

These modules are the following:

- i. Administration and Monitoring console
- ii. Server Distribution
- iii. Node Distribution
- iv. Client Application

These modules can be run from either a shell script or an Ant script. The Ant script is always called "*build.xml*" and it always has a default target called "*run*". To run any of these modules, simply type "*ant*" or "*ant run*" in a command prompt or shell console. The provided shell scripts are named *start<Component>.<ext>* where *Component* is the JPPF component to run (e.g. "Node", "Driver", "Console") and *ext* is the file extension, "bat" for Windows systems, or "sh" for Linux/Unix-like systems.

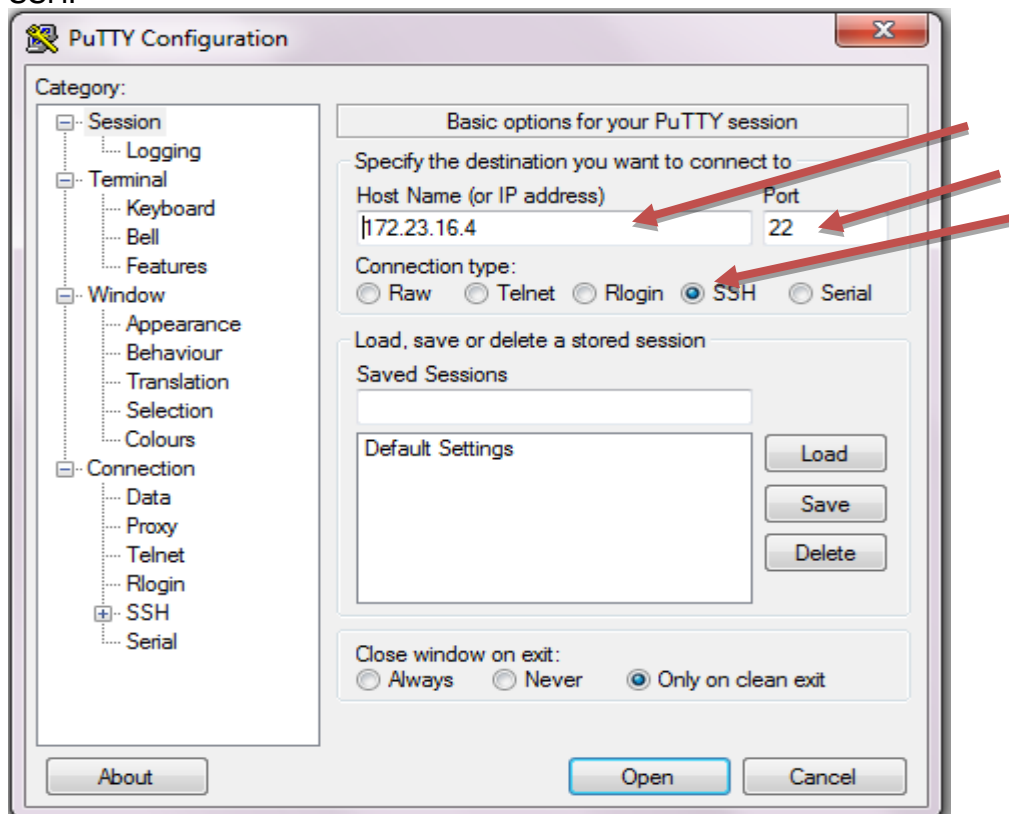
How To Start Server

2.0 How to Start Server

2.1 Creat SSH Session With Server Via Putty

Steps:

- i. Download Putty from <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
- ii. Open Putty.exe.
- iii. Write Host Name (or Ip address), port number and select connection type SSH.



- iv. After that Enter UserName and Password of account created at Server end.

2.2 Run Server

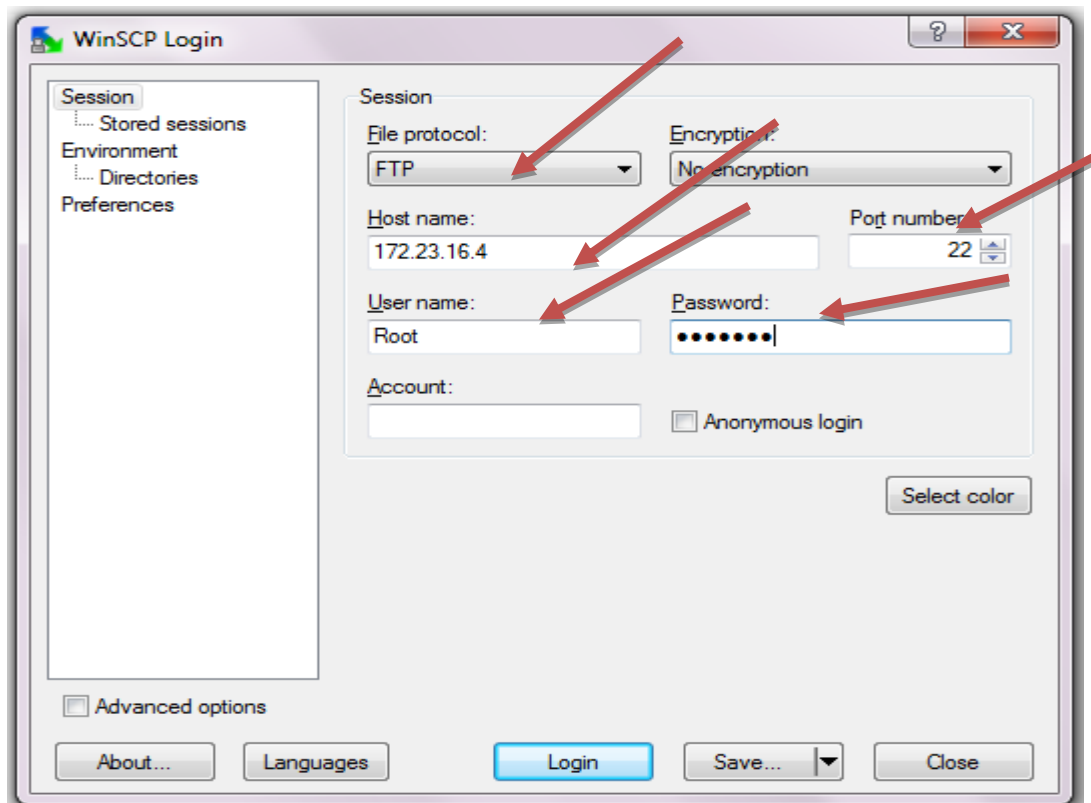
Steps:

- i. As SSH Session is created with server.
- ii. Go to directory where JPPF server is.
- iii. Start server by following commands.
startDriver.bat (for windows)
startDriver.sh (for Unix)
ant (used in both OS)

2.3 Creat FTP Session With Server Via WinSCP

Steps:

- i. Download WinSCP from <http://winscp.net/eng/download.php>
- ii. Open WinSCP.exe.
- iii. Creat New session, select file protocol to FTP, enter Host name (or IP address) and port number and enter UserName and Password of account created at Server end.

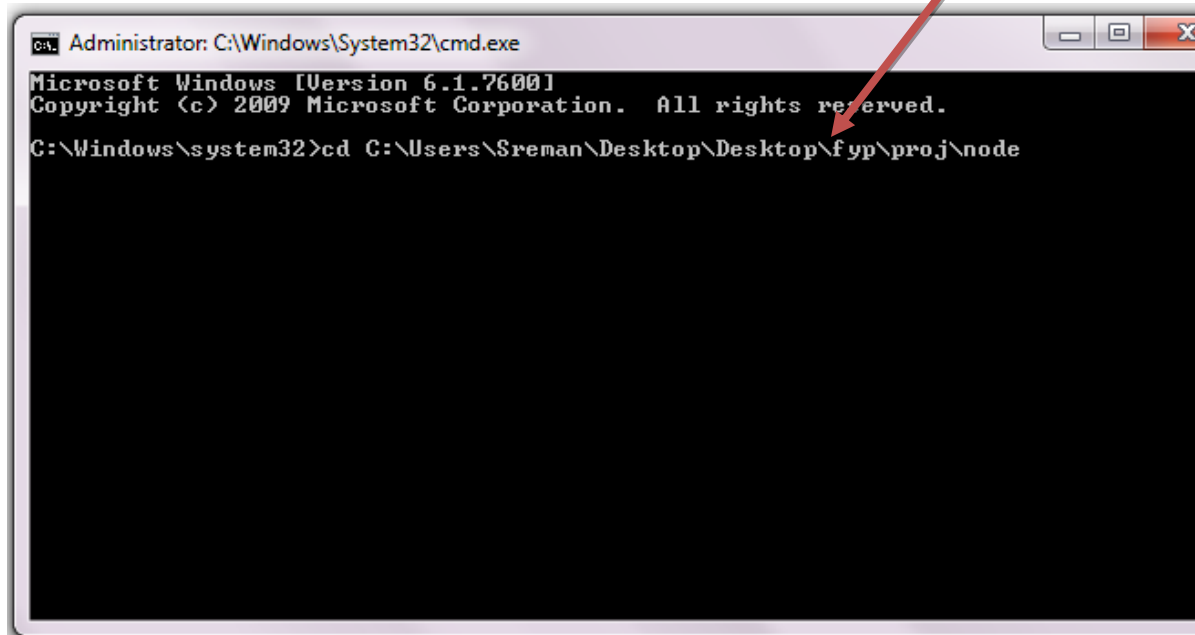


Adding Node

3.0 Adding Node

Steps:

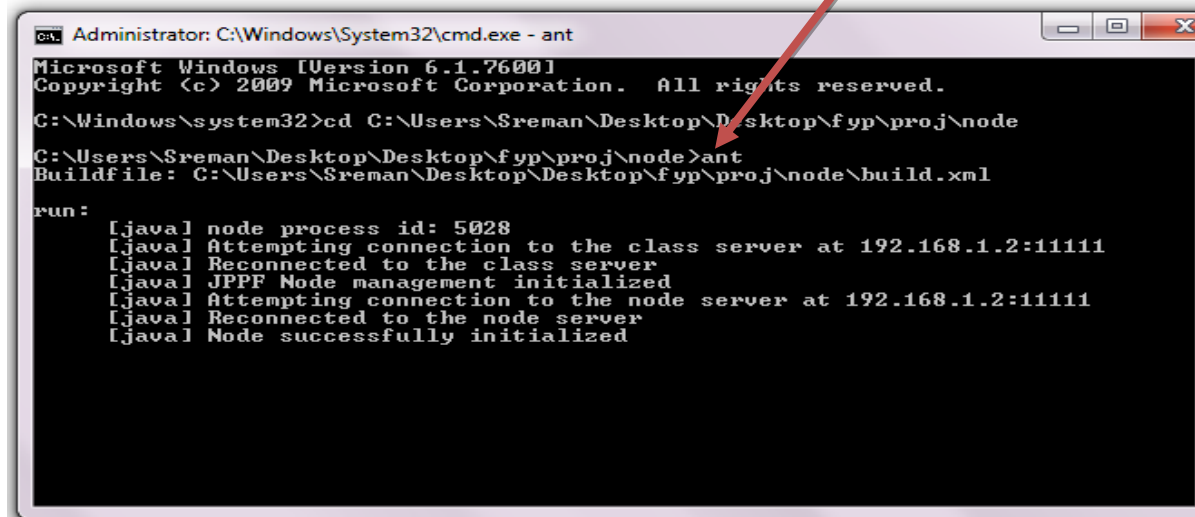
- i. Go to Directory where JPPF node is.



A screenshot of a Windows command prompt window titled "Administrator: C:\Windows\System32\cmd.exe". The window shows the following text: "Microsoft Windows [Version 6.1.7600] Copyright (c) 2009 Microsoft Corporation. All rights reserved. C:\Windows\system32>cd C:\Users\Sreman\Desktop\Desktop\fypp\proj\node". A red arrow points to the path "C:\Users\Sreman\Desktop\Desktop\fypp\proj\node".

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>cd C:\Users\Sreman\Desktop\Desktop\fypp\proj\node
```

- ii. Start node by following commands:
startNode.bat (for Windows)
startNode.sh (for Unix)
ant (works with both)



A screenshot of a Windows command prompt window titled "Administrator: C:\Windows\System32\cmd.exe - ant". The window shows the following text: "Microsoft Windows [Version 6.1.7600] Copyright (c) 2009 Microsoft Corporation. All rights reserved. C:\Windows\system32>cd C:\Users\Sreman\Desktop\Desktop\fypp\proj\node C:\Users\Sreman\Desktop\Desktop\fypp\proj\node>ant Buildfile: C:\Users\Sreman\Desktop\Desktop\fypp\proj\node\build.xml run: [java] node process id: 5028 [java] Attempting connection to the class server at 192.168.1.2:11111 [java] Reconnected to the class server [java] JPPF Node management initialized [java] Attempting connection to the node server at 192.168.1.2:11111 [java] Reconnected to the node server [java] Node successfully initialized". A red arrow points to the path "C:\Users\Sreman\Desktop\Desktop\fypp\proj\node".

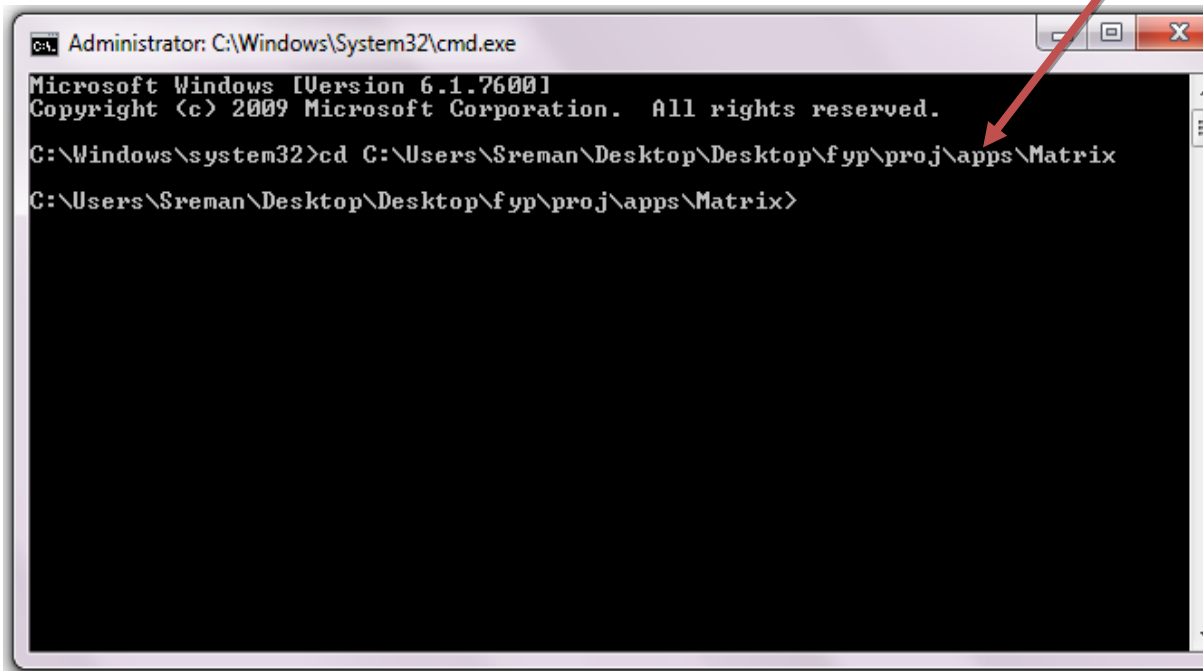
```
Administrator: C:\Windows\System32\cmd.exe - ant
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>cd C:\Users\Sreman\Desktop\Desktop\fypp\proj\node
C:\Users\Sreman\Desktop\Desktop\fypp\proj\node>ant
Buildfile: C:\Users\Sreman\Desktop\Desktop\fypp\proj\node\build.xml
run:
[java] node process id: 5028
[java] Attempting connection to the class server at 192.168.1.2:11111
[java] Reconnected to the class server
[java] JPPF Node management initialized
[java] Attempting connection to the node server at 192.168.1.2:11111
[java] Reconnected to the node server
[java] Node successfully initialized
```

Run Application

4.0 Run Application

Steps:

- i. Check Server is running.
- ii. Check Nodes are added.
- iii. Go to directory where application is.



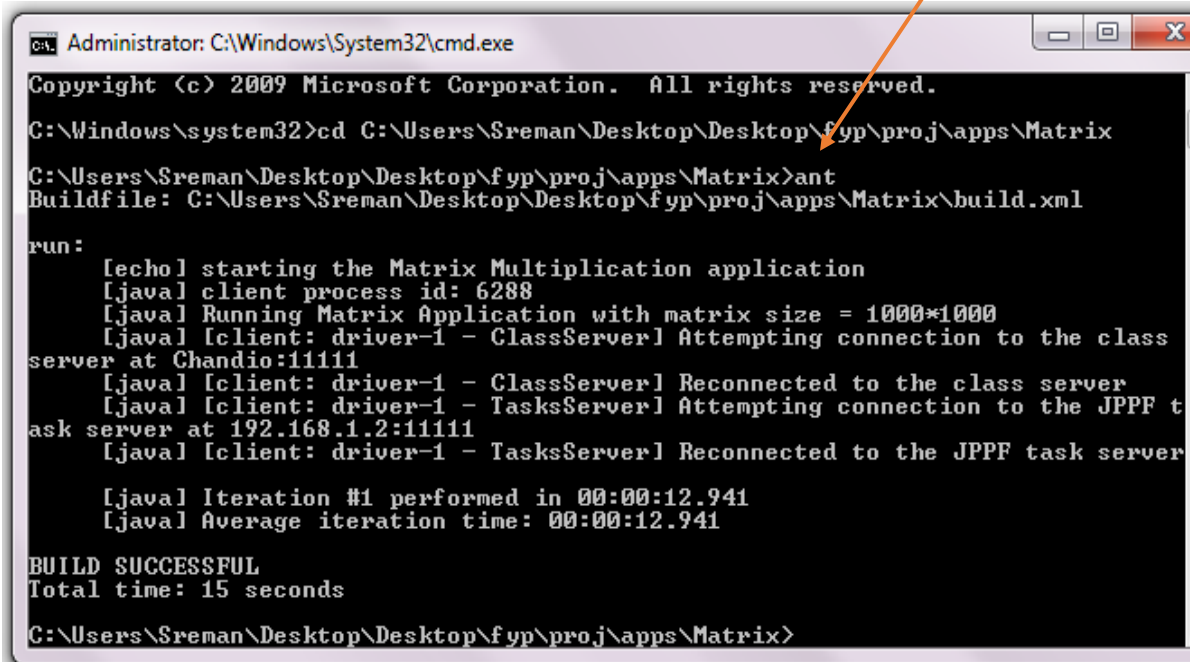
The screenshot shows a Windows command prompt window titled "Administrator: C:\Windows\System32\cmd.exe". The window contains the following text:

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Users\Sreman\Desktop\Desktop\fyproj\apps\Matrix
C:\Users\Sreman\Desktop\Desktop\fyproj\apps\Matrix>
```

A red arrow points to the command prompt window.

- iv. Start application by following commands:
run.bat (for Windows)
run.sh (for Unix)
ant (works with both)



```
Administrator: C:\Windows\System32\cmd.exe
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>cd C:\Users\Sreman\Desktop\Desktop\fypp\proj\apps\Matrix
C:\Users\Sreman\Desktop\Desktop\fypp\proj\apps\Matrix>ant
Buildfile: C:\Users\Sreman\Desktop\Desktop\fypp\proj\apps\Matrix\build.xml

run:
 [echo] starting the Matrix Multiplication application
 [java] client process id: 6288
 [java] Running Matrix Application with matrix size = 1000*1000
 [java] [client: driver-1 - ClassServer] Attempting connection to the class
server at Chandio:11111
 [java] [client: driver-1 - ClassServer] Reconnected to the class server
 [java] [client: driver-1 - TaskServer] Attempting connection to the JPPF t
ask server at 192.168.1.2:11111
 [java] [client: driver-1 - TaskServer] Reconnected to the JPPF task server

 [java] Iteration #1 performed in 00:00:12.941
 [java] Average iteration time: 00:00:12.941

BUILD SUCCESSFUL
Total time: 15 seconds
C:\Users\Sreman\Desktop\Desktop\fypp\proj\apps\Matrix>
```

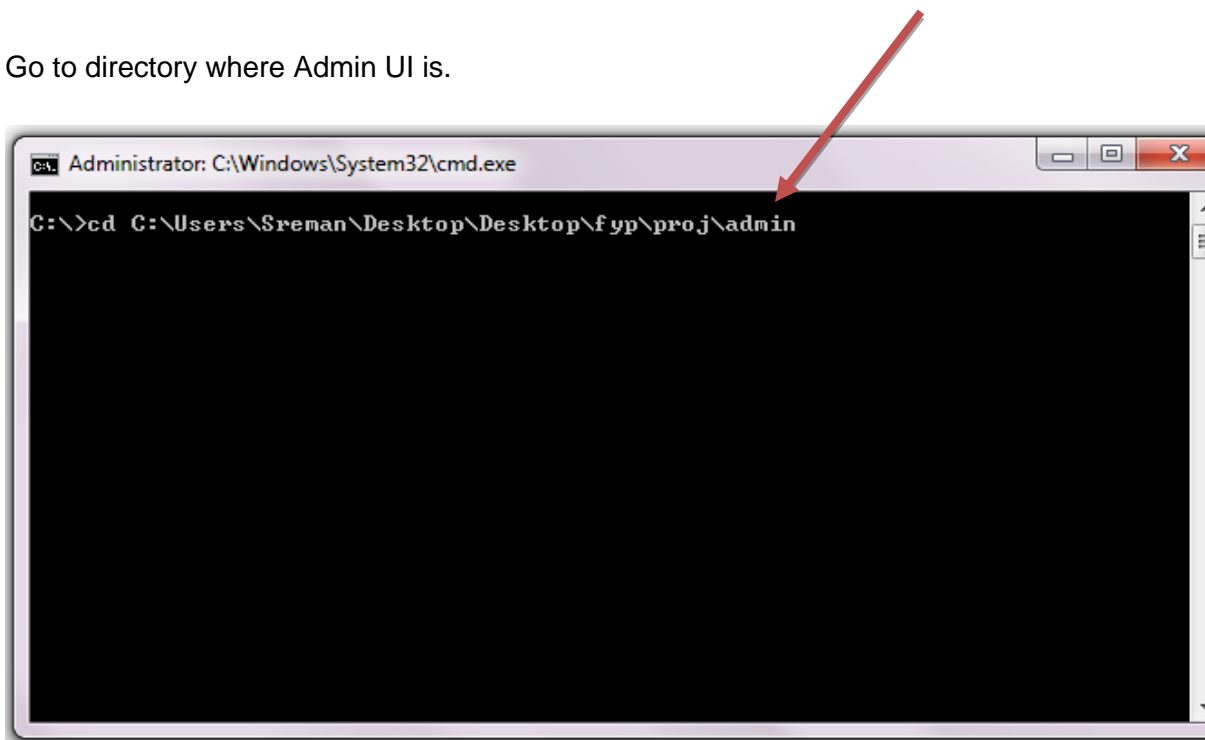
Admin UI

5.0 Admin UI

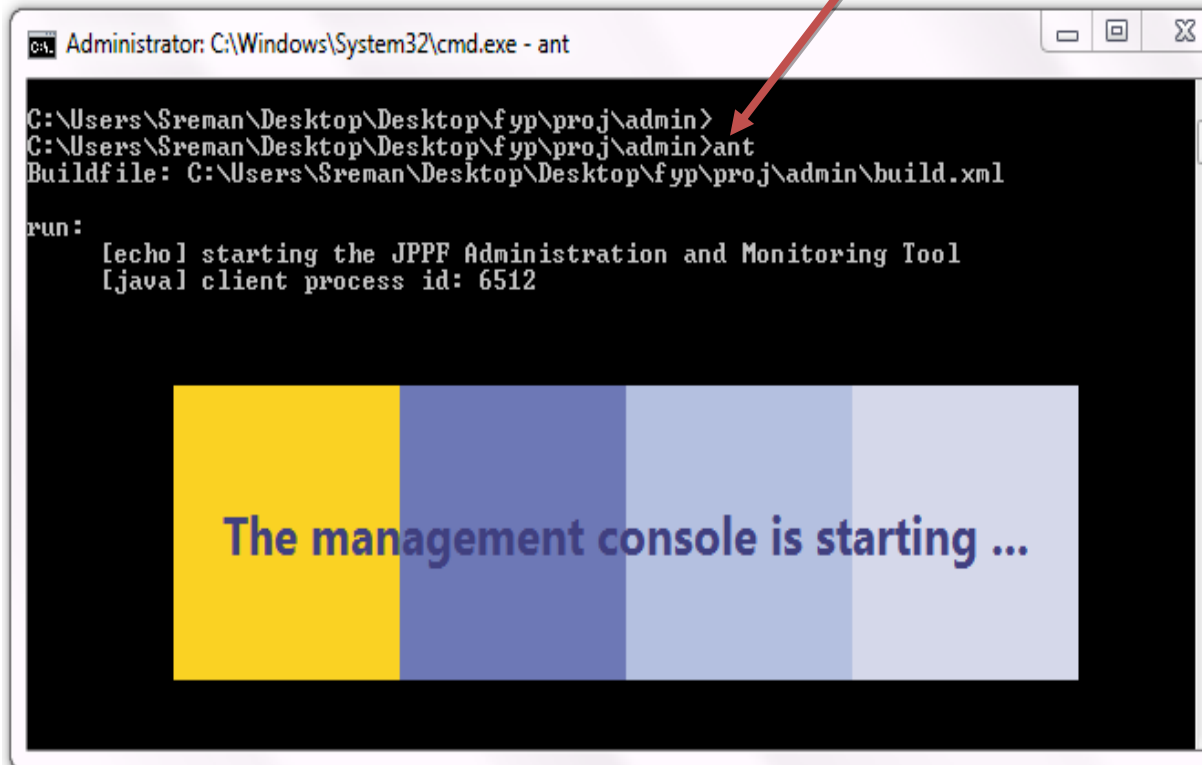
5.1 Starting Admin UI

Steps:

- i. Go to directory where Admin UI is.



- iii. Start Admin UI by following commands:
startconsole.bat (for Windows)
startconsole.sh (for Unix)
ant (works with both)



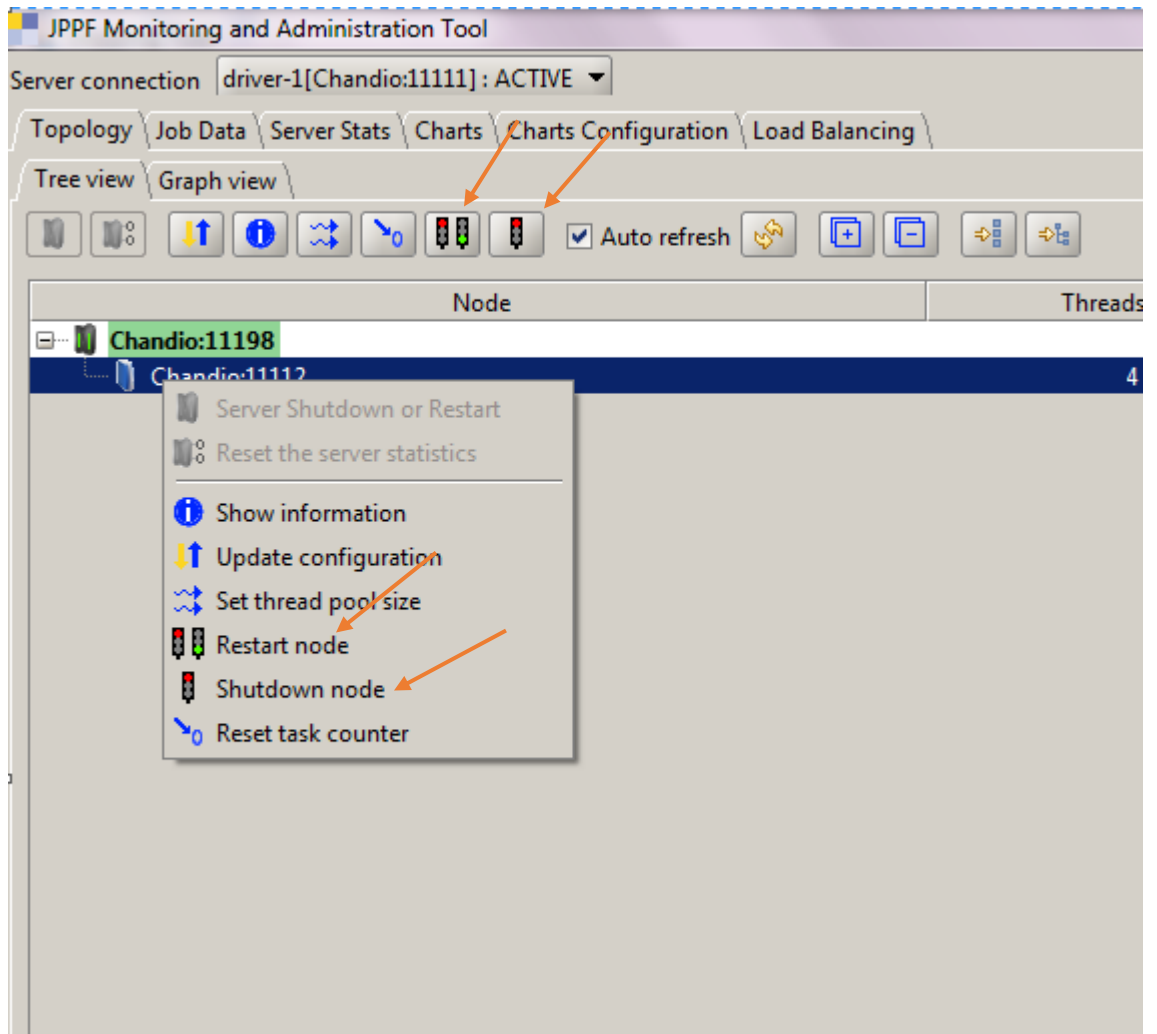
```
Administrator: C:\Windows\System32\cmd.exe - ant
C:\Users\Sreman\Desktop\Desktop\fy\proj\admin>
C:\Users\Sreman\Desktop\Desktop\fy\proj\admin>ant
Buildfile: C:\Users\Sreman\Desktop\Desktop\fy\proj\admin\build.xml
run:
[echo] starting the JPPF Administration and Monitoring Tool
[java] client process id: 6512
```

The management console is starting ...

5.2 Shut Down or Restart Node

Steps:

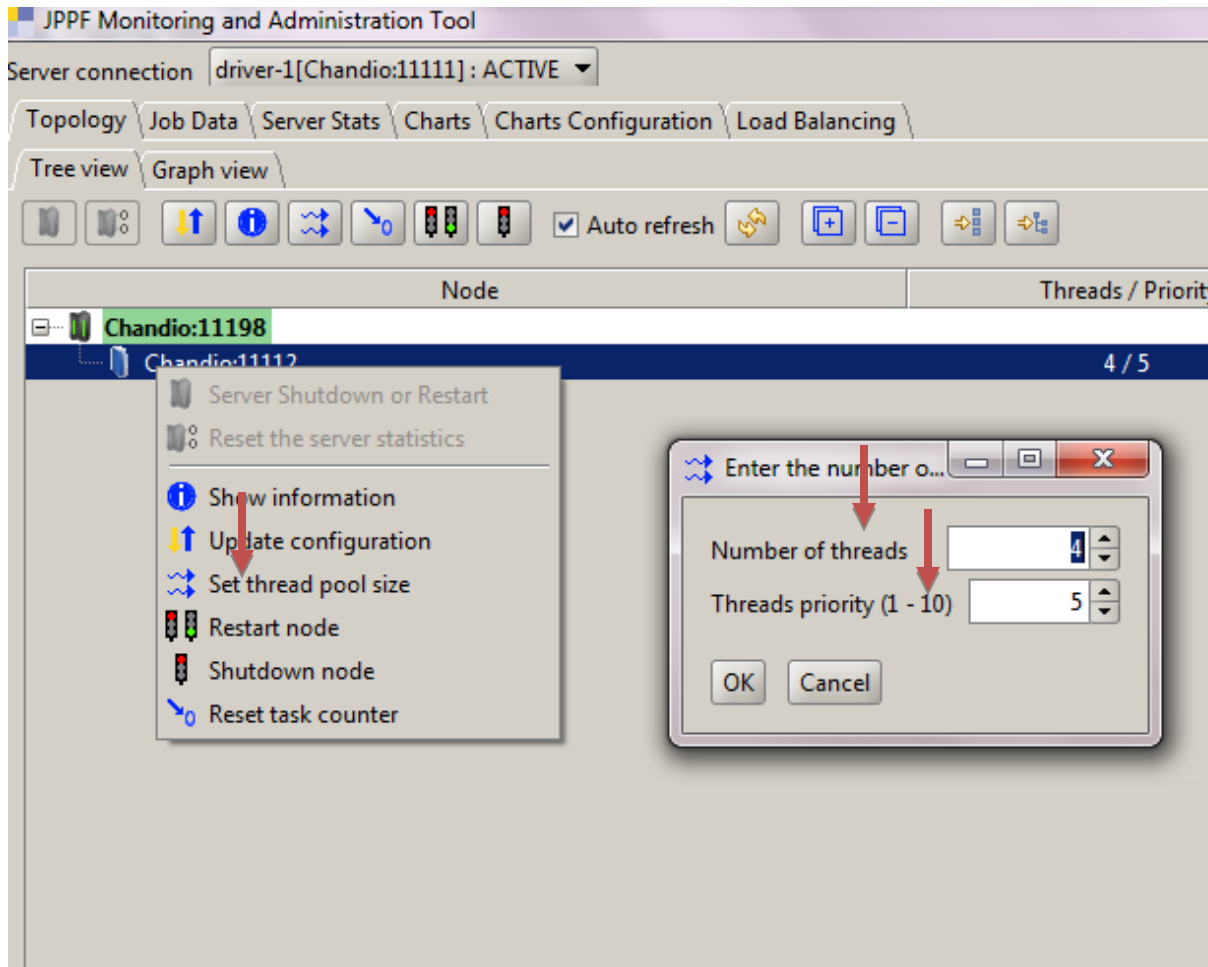
- i. Go to topology tab.
- ii. Right click on node, then select restart or shutdown node.
- iii. Or left click on node just little up from node there is option to shutdown or restart node.



5.3 Set Thread Pool Size and Priority for Node

Steps:

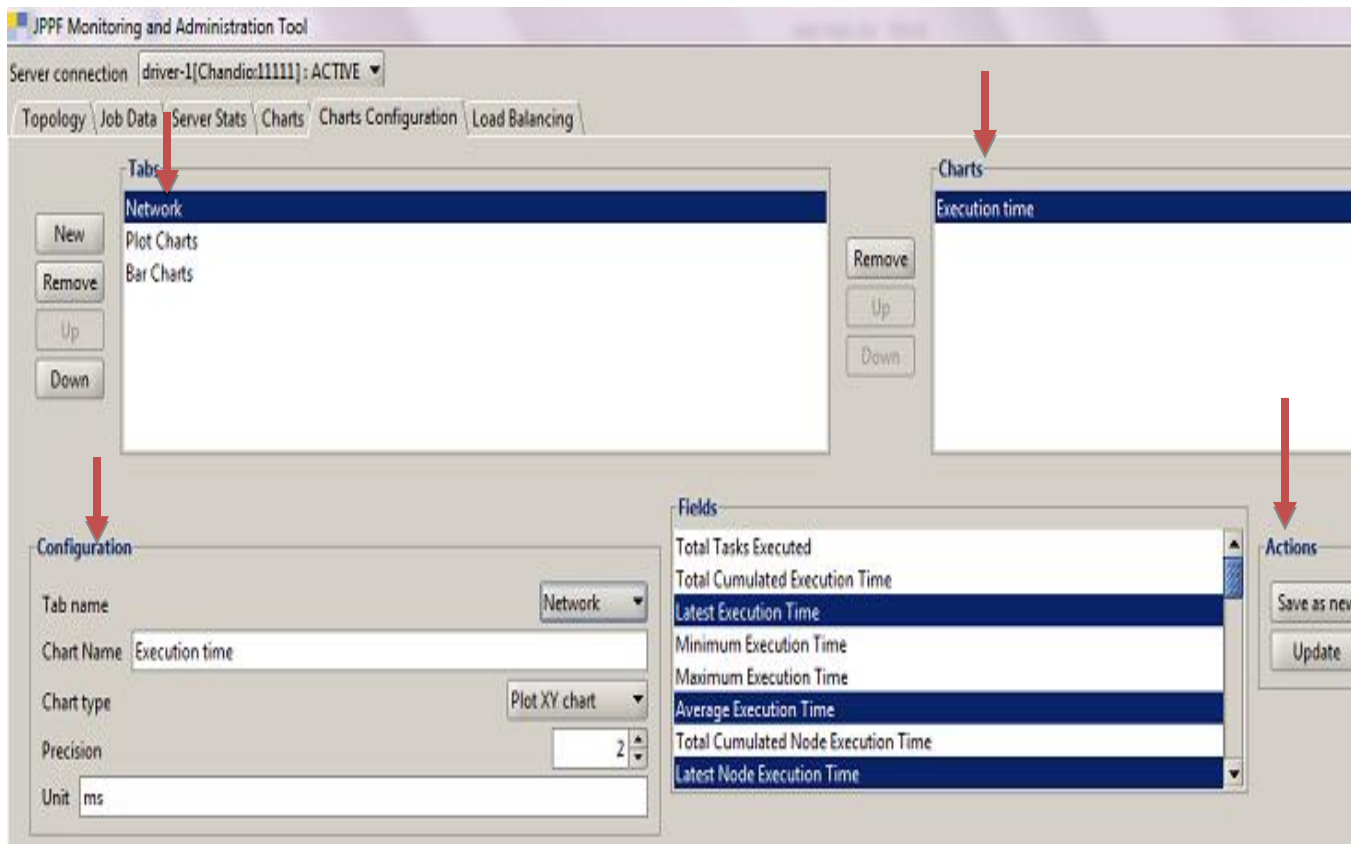
- i. Go to topology tab.
- ii. Right click on any node.
- iii. Select option "set thread pool size".
- iv. Enter number of threads and threads priority.



5.4 Charts Configuration

Steps:

- i. Go to Charts Configuration tab.
- ii. By default there are three tabs select any one of them.
- iii. Then select any chart you want to configure.
- iv. Then go to configuration box and change configuration as you wish.
- v. Then update it or save it as new.

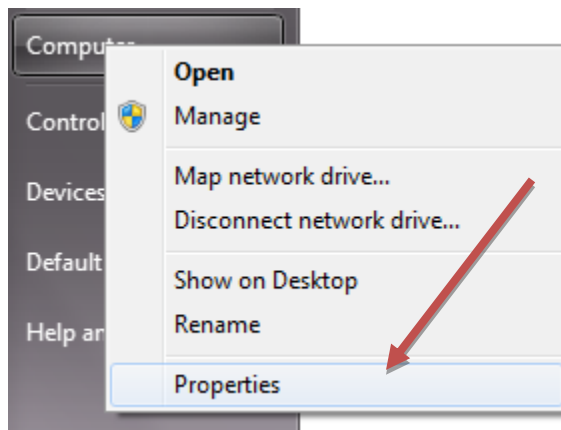


How to Install Ant

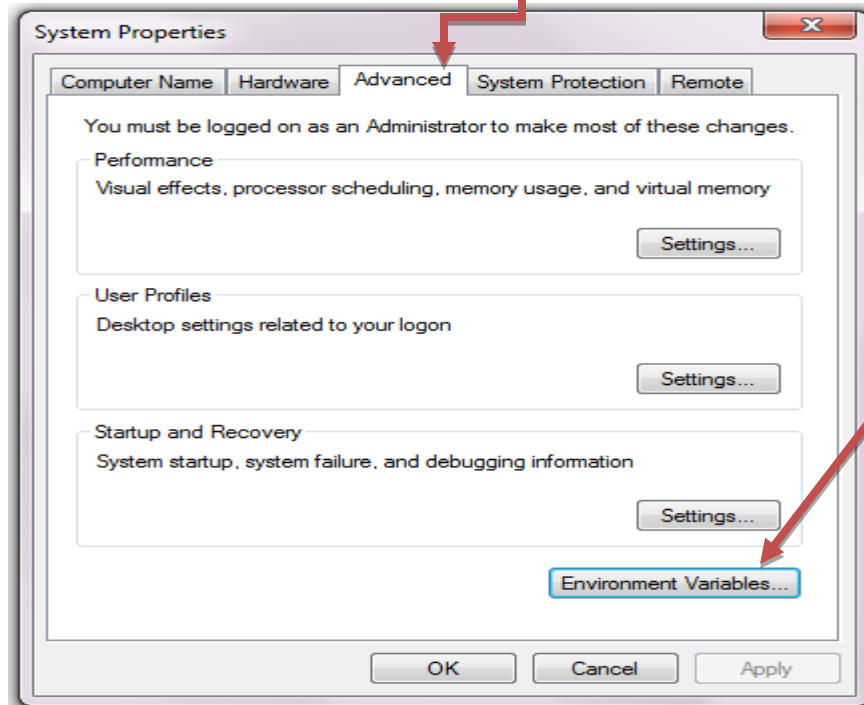
6.0 How to Install Ant

Steps:

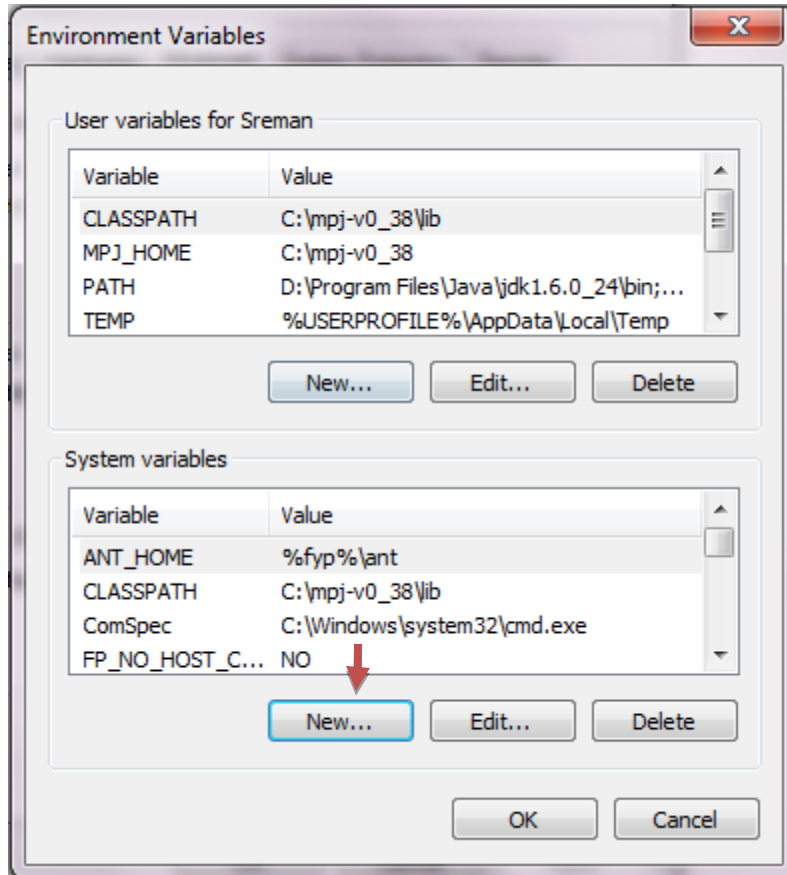
- i. First download ant from <http://ant.apache.org/bindownload.cgi?PreferrPr=ftp://apache.mirrors.pair.com/>
- ii. Right click “My Computer” go to “Properties”.



- iii. Go to “Advanced System Settings”, then go to “Advanced” tab, then Go to “Environment Variables”



- iv. Add new variable in “System variables”



- v. Enter variable name and in value it address (directory where ant is located).

