COP5615 Distributed Systems Fall 2010

Computer and Information Science and Engineering
University of Florida

Project 2

document version 1.0

Date Assigned:	Jan 25, 2010	**Office:**	CSE E327
Due[Campus]:	Feb 05, 2010 local 11:59 pm	**E-mail:**	yyun@cise.ufl.edu
Due[EDGE]:	Feb 07, 2010 edge 11:59 pm	**Contact TA:**	YoungSang Yun

In this project, you are asked to write an application that calculates the power of a square matrix. The objective of the project is to learn the client/server multithreading programming and the associated synchronization problems.

Helpful Guides

Sun Microsystems Guide to Sockets: http://java.sun.com/docs/books/tutorial/networking/sockets/
Multi-threading Tutorial: http://java.sun.com/docs/books/tutorial/essential/concurrency/index.html

Grading Guidelines:

Proper Exception Handling	15%
Clean Termination (runaway processes)	15%
Correct Implementation/Graceful Termination	70%
Total	100%

Guidelines for Project Report: You are required to include a 1-page project report with your submission. It should contain your experience and challenges you faced while doing project 2. You may include any additional comments and suggestions to the TA in this report file. Make sure your report file is named 'report.txt'. Since we will be using UNIX environment to execute your project, we suggest you use a UNIX friendly text editor like pico/nano, vi, etc. to write your report. Make sure the lines in your report does not exceed 80 characters per row.

Submission Guidelines:

- Tar all your source code, config.ini, report.txt, and required output files into a single tar file.
- Name that tar file proj2.tar
- Submit your tar file through e-learning system (lss.at.ufl.edu). Subsequent e-mail submissions will be considered as a late submission.

Regrading Policy: In general, regrading requests will be accepted only for 2 weeks from the date that the initial grading is completed.

CISE Systems Help and Related Issues: The TAs for this class are not responsible for managing CISE department's computing facilities. So if you run into any access problems with your account or if you find some remote machine unresponsive while implementing your project, your best bet will be to contact the CISE admin people directly at *consult@cise.ufl.edu*. Do not email the TAs regarding such issues, the TAs already receive tons of emails.
http://www.cise.ufl.edu/help/ has all the necessary details on CISE clusters and remote machines. You might want to look into this page to figure out what machine to use and where to execute your codes. We require you to write your programs to run correctly in the CISE UNIX environment. **WE WILL NOT TEST ON WINDOWS SYSTEMS**.

1 Introduction

In this project, you are asked to write an application system that calculates $\mathbf{A}^p$ where p is nonnegative integer and $\mathbf{A}$ is the $n \times n$ square matrix. All $\mathbf{A}$, p, and n will be provided in the configuration file. We assume that n is an even number.
The architecture of the system consists of a single **coordinating agent (server)** and several (7) **computing agents (clients)** which are to run on different machines (hosts) connected with a TCP/IP network. The coordinating agent has a square matrix as input and distributes the computations to computing agents. Note that the general meaning of the "server" and "client" is reversed in this project. Thus, to prevent any confusion, we use coordinating agent and computing agents instead of server and client respectively.

2 The Coordinating Agent

The executable file name of the coordinating agent should be "CoordinatingAgent".

2.1 Phase 1

Initially, the coordinating agent must read the "config.ini" that contains the necessary information for both matrix calculation and configuration of remote hosts. The following is an example of the "config.ini". Note that the number of remote node is fixed to 7.

```
#### start of the configuration ######
# any line begin with # should be treated as a comment
# matrix information
# Note that n is even number and p is nonnegative integer
input matrix : A.dat
dimension of matrix : n
exponent : p
# remote node configurations
# node-id hostname port
node1 machine1.cise.ufl.edu 11111
node2 machine2.cise.ufl.edu 11112
```

```
node3 machine3.cise.ufl.edu 11113
node4 machine4.cise.ufl.edu 11114
node5 machine5.cise.ufl.edu 11115
node6 machine6.cise.ufl.edu 11116
node7 machine7.cise.ufl.edu 11117
```

Reading Configuration File
"input matrix" is a simple text file that contains all elements of $\mathbf{A}$. It has n lines and each line has n real numbers separated by any number of spaces. Here's an example file of an "input matrix" when $n = 4$.

```
 0.1  0.2  0.3  0.4
-1.0  3.1  4.4  0.0
 0.3  5.1  3.3  2.1
 4.3 -8.9  1.2  0.01
```

If "input matrix" file does not exist, the coordinating agent should generate a random matrix. In this case, the generated matrix, $\mathbf{A} = [a_{ij}], a_{ij} \in \mathbb{R}^+, 1 \leq i, j \leq n$, should satisfy the following condition.

$$\sum_{j=1}^{n} a_{ij} = 1.0 \quad \forall i \in \{1, \ldots, n\} \tag{1}$$

The matrix with above property is called "stochastic matrix". Note that n is always an even positive integer and p is a nonnegative integer.
After reading "config.ini", the coordinating agent must create 7 threads, called "Communicating Threads" for communication with remote nodes. Each communicating thread should wait for certain amount of time (for example 10 seconds) and then request a connection to the computing agent running on the remote node. When all communicating threads have established connections to the remote nodes, the initial phase of the coordinating agents is finished.

2.2 Phase 2

Before getting into the details of the phase 2, we will introduce the matrix multiplication algorithm, called Strassen algorithm, which is asymptotically faster than the standard matrix multiplication algorithm.

2.2.1 Strassen algorithm for matrix calculation

Let $\mathbf{A}, \mathbf{B}$ be two $n \times n$ square matrix. (Suppose that n is an even integer.) We want to calculate the matrix product $\mathbf{C}$ as

$$\mathbf{C} = \mathbf{AB} \quad \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n \times n} \tag{2}$$

We can partition $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ into equally size block matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix},$$

where $\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in \mathbb{R}^{n/2 \times n/2}$. Then, we define new matrices:

$$\begin{aligned}
\mathbf{M}_1 =& (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 =& (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 =& \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 =& \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 =& (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 =& (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 =& (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}$$

Finally, the $\mathbf{C}$ can be calculated by the following equations.

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}$$

We recommend that you verify the above result.

2.2.2 computation distribution

The coordinating agent will go through $p - 1$ rounds to obtain the final result of $\mathbf{A}^p$ if $p \geq 1$. Special care should be taken in the case of $p = 0, 1$. In kth round, the coordinating agent must calculate $\mathbf{A}^{k+1} = \mathbf{A}\mathbf{A}^k$ with the Strassen algorithm. Note that $\mathbf{A}^k$ is the result of the previous round. As in the description of the Strassen algorithm, each $\mathbf{M}_i$ is the product of two matrices and these products will be calculated in a distributed manner. Each communicating thread must determine two block matrices and send them to the corresponding computing agent. Then communicating threads should wait for the result from the computing agents. When all $\mathbf{M}_i$ are available, the coordinating agent is ready to calculate $\mathbf{A}^{k+1}$. The coordinating agent must write the result of computation in each round into the log file.
Until all communicating threads get the results from the computing agents, the coordinating agent must not go to the next step. Therefore, we will need to synchronize communicating threads. Later we will discuss how to do this. In this description, we do not specify the details of the protocol message between communicating threads and computing agents. You may need to design the structure of your own protocol messages.

2.3 Phase 3

This phase is the final phase of the coordinating agent. Since the coordinating agent might have the final result $\mathbf{A}^p$, the only thing left is to terminate every communicating thread and its corresponding computing agent. This can also be done by the protocol message issued by communicating thread.

3 Barrier Synchronization

It would not be realistic to assume that the matrix multiplications in all computing agents are to be completed all at the same time. Therefore we will need to synchronize the

communicating threads in the coordinating agent. What we will use is called **Barrier Synchronization**, where every thread should wait at a certain synchronization point for the other threads to reach the same point. When all communicating threads reach this point, they are allowed to proceed for the next processing step. In this project, each round is a synchronization point. The parent thread of the coordinating agent should wait for all the communicating threads to receive the result from the computing agents before moving on to the next round.
In java, the synchronization mechanism follows the **monitor** concept through the keyword **synchronized**. To implement the barrier synchronization, we will use two methods in Java which are designed for synchronization: **wait()** and **notifyAll()**. For details of the use of these primitives, refer to the Java documents or manual.

4 Computing Agent

The executable file name for the computing agent must be "ComputingAgent" The job of the computing thread is very simple. It starts with its node id, which is given as command line argument. Then it must read its port number to work on from the "config.ini" file. After the connection between computing agent and coordinating agent is set up, the computing agent should wait for the protocol message. When the computing agent receives two matrices, it needs to calculate the product of the two matrices. The computing agent must take care of the order of two matrices because, in general, $\mathbf{AB} \neq \mathbf{BA}$. The result of the calculation must return to the coordinating agent. The computing agents must write down the received matrices and the product of them in the log file. Also, you can freely use any algorithm to obtain the product of matrices in the computing agent.
When the computing agent receives special protocol message for termination, it must tear down the connection with the communicating thread. The procedure of the termination should be done gracefully.

5 How to Start a Remote Process

To start a process on a remote machine we use the remote login facility **ssh** in Unix systems. It accepts the host name as a parameter and commands to execute. If more than one commands are to be issued, then they must be enclosed by quotation and each command must be separated by semicolons.
To execute the **ssh** command from a Java program, you should use **exec()** primitive of the *Runtime* class.
First, you need to get the project directory (all the files necessary for this project should be in the same directory and start.java will be run in this directory), as follows:

```
String path = System.getProperty("user.dir");
```

Second, you may need to invoke **exec()** methods like this:

```
Runtime.getRuntime().exec("ssh " + host + " cd " + path + "; java
 CoordinatingAgent > node0.log");
```

As in the above example, the output file name of the coordinating agent must be "node0.log". Similarly you can run all 7 computing agents in the start.java as follows.

```
Runtime.getRuntime().exec("ssh " + host + " cd " + path + "; java
 ComputingAgent node1 > node1.log");
```