CS5523 Operating System

A Distributed Matrix Multiplication Application

Name: Sakhawat Hossain Saimon
Instructor: Dr. Mimi Xie

05/13/2020

Abstract

In this project, a Java application is designed to tackle the problem of multiplying large matrices using a distributed system of machines. Matrix multiplication is known to be computationally intensive, but there is scope for parallelization, which is exploited in the application. The application is logically partitioned into two core modules Master and Worker, where the first module is charged with deploying matrix multiplication jobs to one or more instances of the later, and combining the results.

1 Introduction

1.1 Problem Definition

The number of multiplication operations required to multiply two matrices grows quickly with the dimensions and number of matrices. There are two variants of this problem:

1. Multiplying two large matrices: without loss of generality, let the number of rows and columns in A and B be n. The resultant matrix also has n rows and columns, with a total of n^2 cells. Calculating each cell of the matrix requires n multiplications, and the total number of multiplications is $O(n^3)$.
2. Multiplying a long chain of matrices, $A_1, A_2, A_3, ...A_n$. If each matrix is $n \times n$, then the total number of multiplications is $O(n^4)$.

For this project, we only consider the first variant.

1.2 Goal

Unlike many computational problems, there has been little improvement in the upper bound of operations for matrix multiplication. The simple approach mentioned above requires $O(n^3)$ running time, while the best possible upper bound achieved is $O(n^{2.376})$ [1]. However, the independent nature of the multiplications immediately suggests that it can be exploited by a distributed computation approach.

It should be noted that breaking apart the input and using a group of machines to solve subproblems is not a novel idea. Apache Hadoop [3] is one of many MapReduce libraries available to programmers that allow using distributed systems for solving computationally demanding problems.

2 Overview

In this project, we present a distributed, machine-independent application for simple matrix multiplication tasks. Given k machines, we define the running time of matrix multiplication as follows:

$$T_d + \frac{O(n^3)}{\min{(k, n^2)}} + T_c$$

Where T_d is the time required to divide a single matrix multiplication task as n^2 independent multiplications, which can be assigned to k different machines. Each machine computes the multiplications assigned to it in parallel with other machines. The results are then collected and combined in T_c time. However, there is a limit to this speedup, as evident by the denominator $\min{(k, n^2)}$. If there are exactly than n^2 machines in the system, then each machine is tasked with computing a single cell of the resultant matrix. If there are more than n^2 machines in the system, then the other machines sit idle. Regardless of the limit, if there are n^2 machines, and if T_d and T_c are small compared to n, then the time to multiply two $n \times n$ matrices effectively becomes $O(n)$, which is more than an order of magnitude faster than the fastest algorithm that does not make use of parallelization.

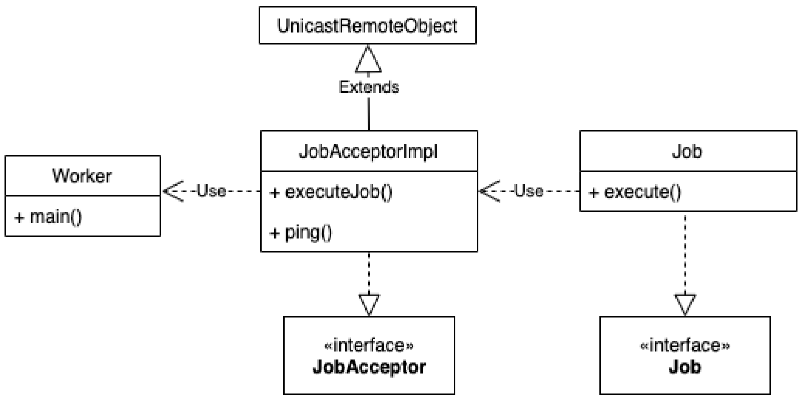


Figure 2.1: Class Diagram of the Worker Module

2.1 Technologies:

The application is built using Java, which supports generating machine-independent code. The key technologies used are as follows:

1. **Java RMI:** The Remote Method Invocation (RMI) framework provides a convenient way of shipping both code and data from clients to RMI servers for execution [2].
2. **Sockets:** Used for initial communication
3. **Threads:** Used for running tasks in background.

2.2 System Architecture

The system consists of two core modules: Master and Worker. Each system can decide whether to act as a server or as a client, by launching the respective modules. The UI is implemented in a terminal/command-line window. The project is designed for local-area-network address, which can be extended to support communication over the world-wide-web by adding support for port forwarding.

2.2.1 Worker Module

The Worker module hosts an RMI server, and carries out matrix multiplication tasks handed by master. A simplified approach would be to have the matrix multiplication code under the worker module, and exchange only numeric values between Master and Workers. However, this is against the interest of scalability, as the application can not be expanded in the future to tackle tasks other than matrix multiplication. Having this in mind, we design the Worker module in such a way that it contains no code for matrix multiplication. The Worker provides the framework for accepting Java objects that contain both code and data, unmarshalling and executing code on the data, and returning the results.

This scheme, no matter how scalable, has security issues. Malicious clients can pass harmful code to the client which is run without checking the code for irregularities. It is therefore desirable to make sure that the Worker provides its services to only trusted clients. This is achieved using initial socket communication, and the user must explicitly enter a Master node's ID at the Worker end. The Worker then sends its own RMI binding address to Master, which is used for distributed computing. Figure 2.1 provides an overview of the core classes that make up the Worker module.

2.2.2 Master Module

The Master module hosts an RMI client, and contains the code for marshalling matrix multiplication code and data (figure 2.2). At startup, Master presents its communication

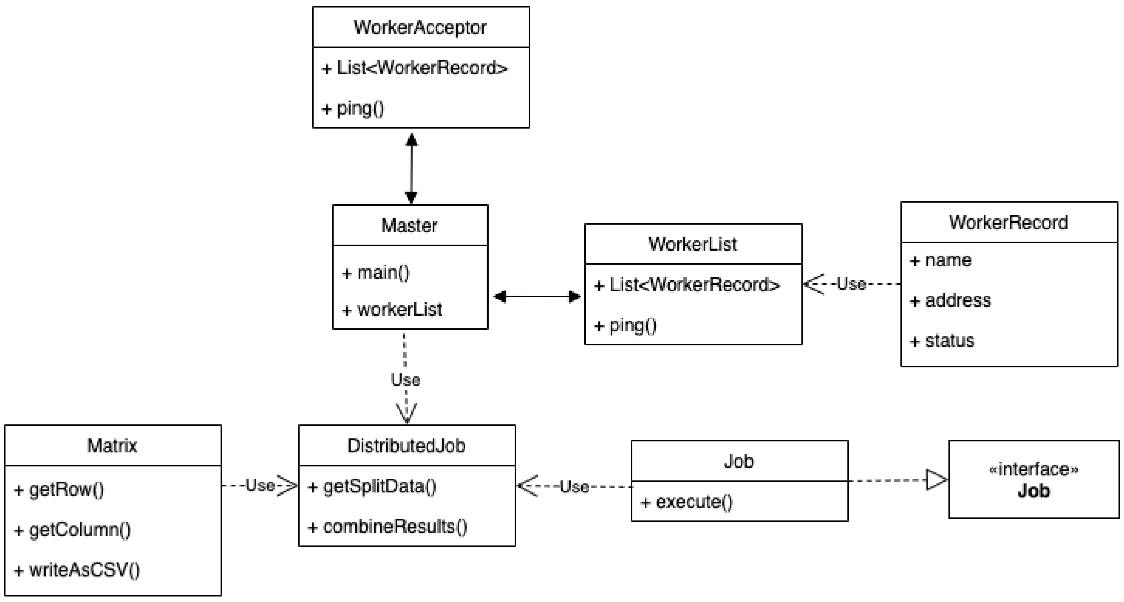


Figure 2.2: Class Diagram of the Master Module

ID to the user, which can be entered at any Worker module to sign up for work from Master. Master also starts a separate thread for accepting initial socket connections from workers, and records the worker's credentials into a component named `WorkerList`. Each record in the `WorkerList` module contains the name, address, and status of each worker.

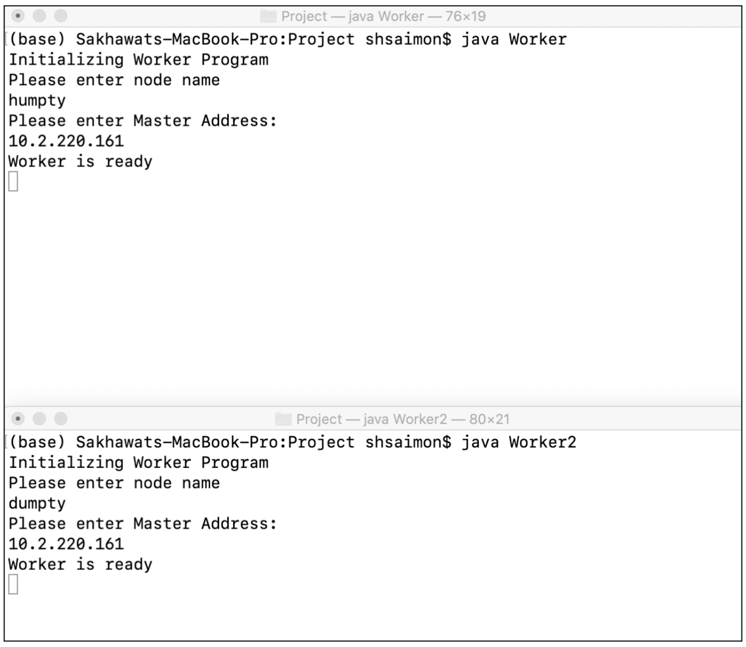
The classes `DistributedJob` and `Matrix` contain the code for reading two matrices from plaintext file, splitting the matrices into n^2 jobs, job execution and combining results. The `WorkerList` class accepts an instance of the `DistributedJob` class and distributes the split jobs to each of the registered Workers evenly. It then collects the results and combines them using the code in `DistributedJob`. While we were unable to implement complete dependency-inversion due to lack of time, the architecture provides ample scope for adding it. It is possible for users to deploy their own custom-designed distributed tasks by adding a few more interfaces to the code.

The `WorkerList` class contains the code for managing the connection to workers. It also contains code for detecting servers that are not-responding and removing them from its list. However, the detection contains a bug which fails to detect a connection, despite being wrapped in broad-scope exception handling code. We were unable to fix this issue within the deadline.

3 Experiments

We use one Master instance and two Worker instances running on a single machine as the testing environment due to lack of resources. This also limits us to testing the correctness of the program and its load-distribution capacity. We maintain that there is scope for broad testing of the effective running time with more nodes. We also make sure that RMI registry is running on the machine.

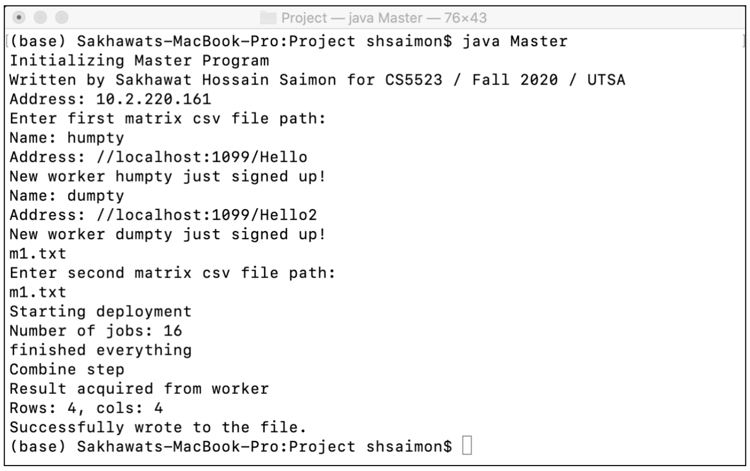
3.1 Test Case 1



```
(base) Sakhawats-MacBook-Pro:Project shsaimon$ java Worker
Initializing Worker Program
Please enter node name
humpty
Please enter Master Address:
10.2.220.161
Worker is ready
```

```
(base) Sakhawats-MacBook-Pro:Project shsaimon$ java Worker2
Initializing Worker Program
Please enter node name
dumpty
Please enter Master Address:
10.2.220.161
Worker is ready
```

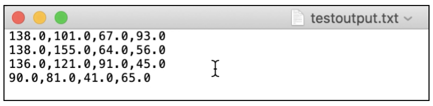
Figure 3.1: Experiment 1 - Worker nodes before receiving job



```
(base) Sakhawats-MacBook-Pro:Project shsaimon$ java Master
Initializing Master Program
Written by Sakhawat Hossain Saimon for CS5523 / Fall 2020 / UTSA
Address: 10.2.220.161
Enter first matrix csv file path:
Name: humpty
Address: //localhost:1099/Hello
New worker humpty just signed up!
Name: dumpty
Address: //localhost:1099/Hello2
New worker dumpty just signed up!
m1.txt
Enter second matrix csv file path:
m1.txt
Starting deployment
Number of jobs: 16
finished everything
Combine step
Result acquired from worker
Rows: 4, cols: 4
Successfully wrote to the file.
(base) Sakhawats-MacBook-Pro:Project shsaimon$
```

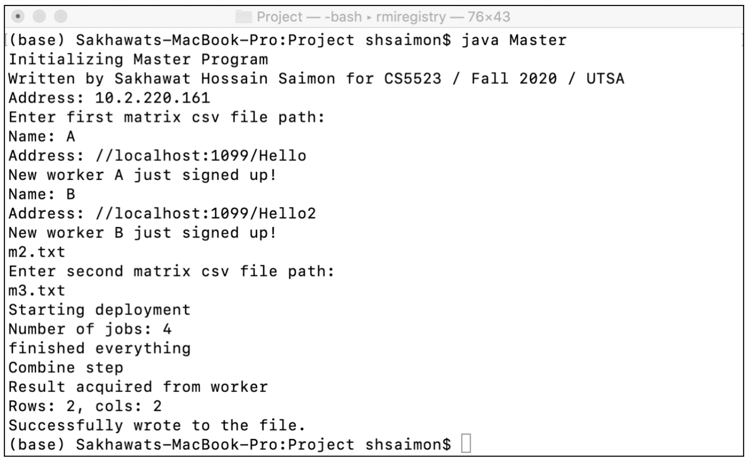
Figure 3.2: Experiment 1 - Master Node

A 4×4 matrix was multiplied to itself. Figure 3.1 shows Workers signing up to master. Figure 3.2 shows Master accepting and acknowledging workers, reading matrix from comma-separated value file and assigning job to workers. Finally, figure 3.6 shows the correct calculated results saved to file by Master.



```
138.0,101.0,67.0,93.0
138.0,155.0,64.0,56.0
136.0,121.0,91.0,45.0
90.0,81.0,41.0,65.0
```

Figure 3.3: Experiment 1 - Result written to file

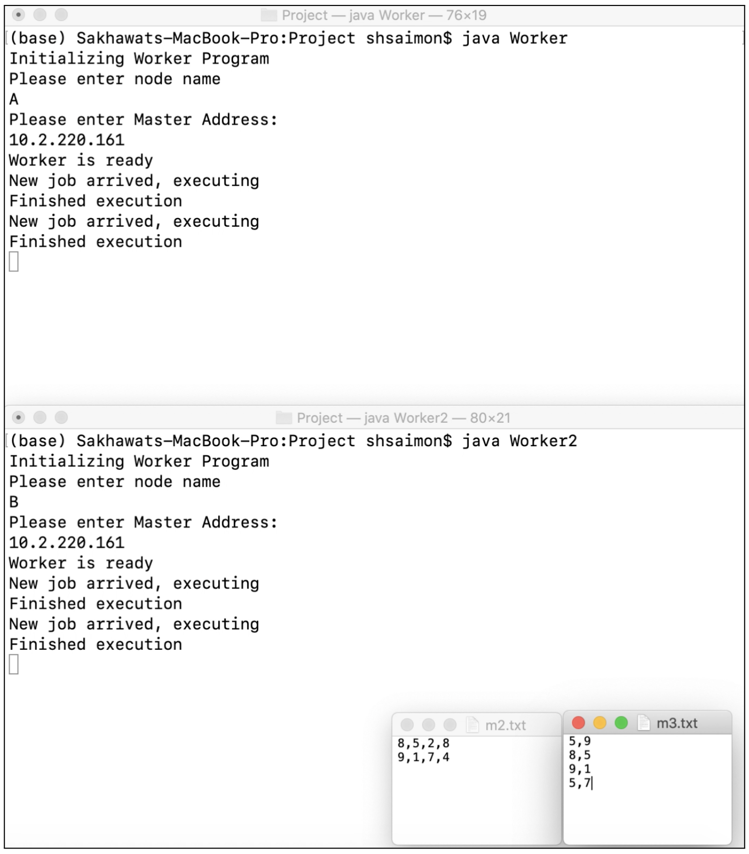


Project — -bash ▸ rmiregistry — 76×43

```
(base) Sakhawats-MacBook-Pro:Project shsaimon$ java Master
Initializing Master Program
Written by Sakhawat Hossain Saimon for CS5523 / Fall 2020 / UTSA
Address: 10.2.220.161
Enter first matrix csv file path:
Name: A
Address: //localhost:1099/Hello
New worker A just signed up!
Name: B
Address: //localhost:1099/Hello2
New worker B just signed up!
m2.txt
Enter second matrix csv file path:
m3.txt
Starting deployment
Number of jobs: 4
finished everything
Combine step
Result acquired from worker
Rows: 2, cols: 2
Successfully wrote to the file.
(base) Sakhawats-MacBook-Pro:Project shsaimon$
```

Figure 3.4: Experiment 2 - Master Node

3.2 Test Case 2



Project — java Worker — 76×19

```
(base) Sakhawats-MacBook-Pro:Project shsaimon$ java Worker
Initializing Worker Program
Please enter node name
A
Please enter Master Address:
10.2.220.161
Worker is ready
New job arrived, executing
Finished execution
New job arrived, executing
Finished execution
```

Project — java Worker2 — 80×21

```
(base) Sakhawats-MacBook-Pro:Project shsaimon$ java Worker2
Initializing Worker Program
Please enter node name
B
Please enter Master Address:
10.2.220.161
Worker is ready
New job arrived, executing
Finished execution
New job arrived, executing
Finished execution
```

m2.txt

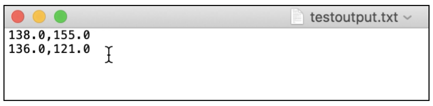
```
8,5,2,8
9,1,7,4
```

m3.txt

```
5,9
8,5
9,1
5,7
```

Figure 3.5: Experiment 2 - Worker nodes after receiving job

A 2×4 matrix was multiplied to another 4×2 matrix. Figure 3.4 shows Master accepting and acknowledging workers, reading matrix from comma-separated value file and assigning job to workers. Figure 3.5 shows each worker receive 2 of the 4 total cell-computing jobs. Finally, figure 3.3 show the correct calculated results saved to file by Master.

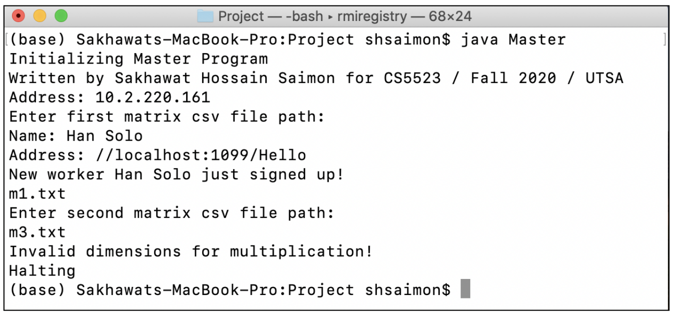


```
138.0,155.0
136.0,121.0
```

Figure 3.6: Experiment 2 - Result written to file

3.3 Test Case 3

A 4×4 matrix was multiplied to a 4×2 matrix. Master is able to detect the incorrect dimensions and stop jobs from being deployed.



```
(base) Sakhawats-MacBook-Pro:Project shsaimon$ java Master
Initializing Master Program
Written by Sakhawat Hossain Saimon for CS5523 / Fall 2020 / UTSA
Address: 10.2.220.161
Enter first matrix csv file path:
Name: Han Solo
Address: //localhost:1099/Hello
New worker Han Solo just signed up!
m1.txt
Enter second matrix csv file path:
m3.txt
Invalid dimensions for multiplication!
Halting
(base) Sakhawats-MacBook-Pro:Project shsaimon$
```

Figure 3.7: Experiment 3: Detecting Invalid Matrix Dimensions

4 Discussion and Conclusions

In this project, we designed an application that serves as a working demo and a lightweight, stripped-down alternative to other MapReduce applications. The experiments demonstrate that the application produces correct results.

Further work can be done to implement custom job deployment options from user-written code, which was one of the major goals of the architecture. The interface can be ported to GUI, and job queues can be introduced to support batching multiple heterogeneous jobs.

Team Member Contribution

Sakhawat Hossain Saimon: 100%

References

[1] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.

[2] E. Pitt and K. McNiff. *Java. rmi: The remote method invocation guide*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.