CS3006 Parallel and Distributed Computing
Project
Emulating Map Reduce using MPI

Submission Instructions:

- **You are required to submit .c files as well as any input files you create in a zip folder**
- **Strict plagiarism policy applies to the code submitted**
- **Viva can be taken for any suspicious submission**
- **Non-running code will be awarded zero marks**

Background

The aim of the project is to utilize MPI in C to emulate the Map Reduce framework. Map Reduce works by having a Nodemanager which accepts a job and then splits the input to the mappers, and mappers inform the Nodemanager after the completion of their jobs. Nodemanager will shuffle the output of the mappers and send them to the reducers. You can find an architecture diagram of hadoop in Figure 1.

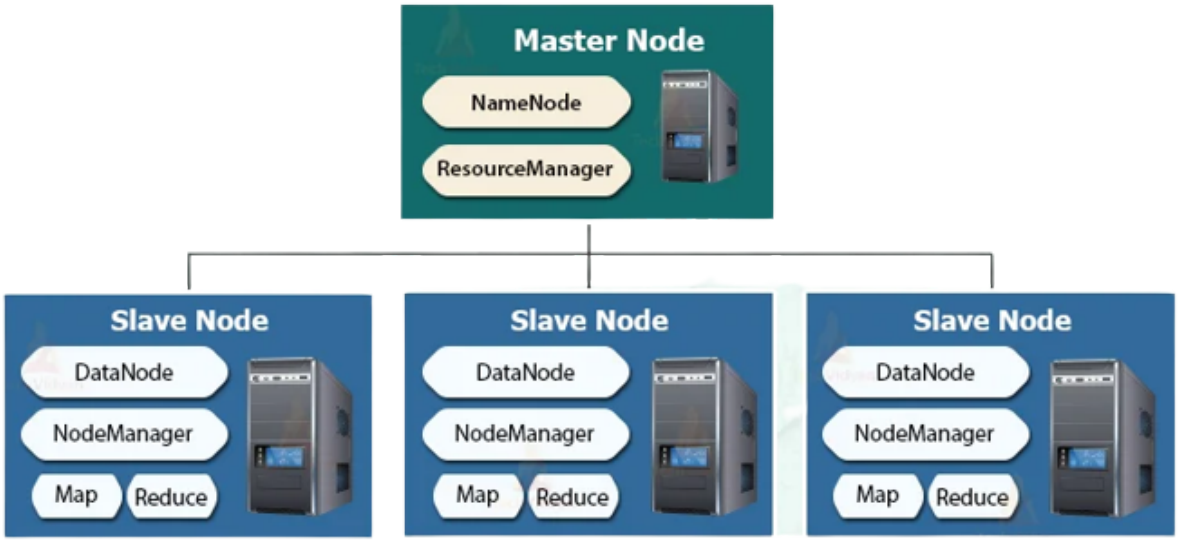

Figure 1. Hadoop architecture

This process is to be emulated using MPI. In MPI, we can also have a master slave configuration by assigning one process the task of being the master and the rest as slaves. In this regard, we can use MPI to have a master node, which can be let's say process 0. Now the rest of the processes can act as slave nodes that can be treated as either mappers or reducers. Following are some examples using 8 processes for the sake of understanding. The assignment of processes to mappers and reducers for the project will be dynamic. The details are available in next section.

1. Process 0 as master. Processes 1-5 as Mappers, Processes 6,7 as Reducers.
2. Process 0 as master, Processes 1-7 all first act as Mappers, then randomly chosen processes act as reducers depending on how many are actually required.

Project Requirements

You are required to implement the working of Matrix Multiplication in the Map Reduce framework, which will be emulated using MPI. Therefore, you are also required to choose the correct key value pairs for the implementation of your program as per your logic.

a. Create square random matrices of sizes greater than or equal 2^4 and save them to files. You can write a separate program for this operation.
b. The master process will assign the task of splitting the input to the mappers.
c. The mappers will split the input into key, value pairs and inform the master process after they have completed their job.
d. The master process will shuffle this output similarly to what is done in MapReduce and then assign the reduce jobs to the reducer(s).
e. The reducer(s) will do their work on the input passed to them and then write out key, value pairs to a file.
f. The reducer(s) will notify the master after they have completed their job.
g. Then, the master process will convert these key, value pairs to a matrix form and write out the result to a file.

Execution Details

Whenever a program is executed, the filename of the input files is passed as command line arguments.

This assignment of processes as mappers and reducers should be dynamic and dependent on the number of processes the program is executed on.

Expected output

At the start, master will print it's own information:
Master with process_id <process_num> running on <machine_name>

Master will print the following message whenever it assigns a task:
Task <Map/Reduce> assigned to process <process_num>

Each time a process is assigned a task, it must print out it's process rank, the machine it is running on and the task (map or reduce) it is assigned. You can utilize the *MPI_Get_processor_name()* function to print the name of the machine. Please refer to this link for more information.

Hence, each mapper or reducer will print the following message whenever they are assigned a task:
Process <process_num> received task <Map/Reduce> on <machine_name>

Master will print the following message whenever it receives the status of completion of a task:
Process <process_num> has completed task <Map/Reduce>

Master will inform the user when the entire job has been completed.

Master will then compare the matrix multiplication output with the output of serial matrix multiplication program on the same input matrices. It will then print if the two outputs are same or not.

A sample output for 8 processes has been attached below in Figure 2. Note that the ordering of processes in this output is consistent, it is possible that this order might be different. However, as should be evident through how Map Reduce works, no Reduce process should start before a Map process has finished its task.

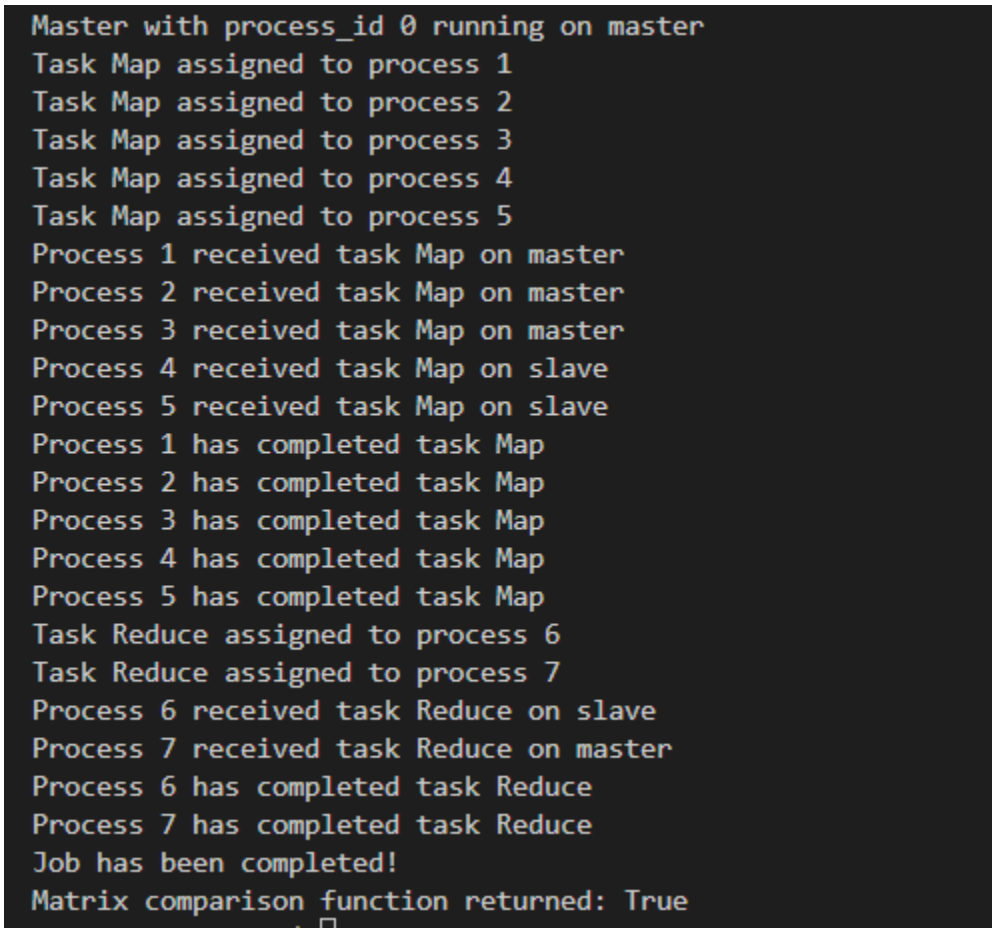

Figure 2. Sample output

Note: The use of Beowulf Cluster is a must for the project. The machine names printed by each process should clearly indicate that there are multiple machines that the program is executing on. Any project demonstrated without the use of Beowulf Cluster will be awarded 0 marks.