System Design

Team members:
Jia Li (li001011)
Kai Wang (wang8739)

Assumption
• There will be a single job at a time.
• All communications in this project are synchronous.
• There will be no faulty node during job execution.
• Every time the master server changes its mode, all nodes need to restart.
• Every time a compute node changes its parameters, all nodes and server need to restart.

Overview
To implement MapReduce, we need a client to send input directory including all input files to the server. The server creates a map task for each input file and sends tasks to compute nodes based on different scheduling policies. Every compute node creates a thread for each task. For each map task, a compute node produce an intermedia file containing the result, which is the input file name and its corresponding sentiment points. When finishing a map task, a compute node would send a notice back to the server. The server waiting while compute nodes do their jobs. After all map task finish, the server sends a list of all the intermediate files to a random compute node to do the sorting task. This compute node would create a thread for sort task and notice the server the result file name after it is done. Then, the server sends the file name and used time back to the client, the whole job is done.

There are 3 components in the system:

Client:

The client sends a job to the server. When the job is done, the client will get a file contains an ordered list of filenames based on the sentiment score, and the elapsed time is printed on the screen.

MasterServer:

The server receives the job submitted by the client. It splits the job into multiple tasks and assigns each task to a compute node. If some file does not exist, the whole task exit and return error message “Error, some file does not exist.”. The server print the list of filenames of completed map task. When the job is completed, the server prints the elapsed time and returns a result filename to the client.

MasterServer interfaces including:

Result sendTask(1: list<string> input_list): used for client to send input file list to server and get a result containing the output filename and elapsed time.

ServerData registerNode(1: string ip, 2: i32 port): used for compute nodes to register themselves.

void noticeFinishedMap(1: string resultFilename): used for compute nodes to indicate the finish of mapping task

void noticeFinishedSort(1: string resultFilename): used for compute nodes to indicate the finish of sorting task

ComputeNode:

The compute nodes execute tasks (either map or sort) sent to them by the server. Compute node interfaces including:

bool mapTask(1: string filename)

void sortTask(1: list<string> filenames)

These interfaces invoke the following two thread class:

MapTask: It has two private static fields to accelerate speed: PosLib and NegLib, which has all positive and negative words to avoid repeated construction of HashSet. To compute the sentiment score of each input file, we first read in context only of “-a-zA-Z”, and then split the tokens with “--”. Since letters after ’ are usually short and meaningless, thus this won’t cause problems in sentiment scoring.

SortTask: In order to sorts all the intermediate files, it firstly saves pairs of filename and score into a map. Then we override the compare method to sort the pairs.