Lei Huang; huan1397; 5330737
Yihan Zhou; zhou1298; 5547186

Design document - describe each component

For this project, we built a distributed computing system to process map-reduce like jobs (sentiment score rank program). We use Java to do the development and use Thrift framework to build communications between different components. There are three major components in the system: Client, Server and Compute Nodes.

1) Client: Client is used to submit the job to the system by calling the particular RPC - submitJob() defined on the Server side. Client is aware of the IP address and open port of the Server to build the connection. The provided parameter by Client is the name of the directory which contains all the input files. Client submits the job with the designated directory info to the server and the return values is the name of the final output file along with the execution elapsed time.

2) Server: Server is used to accept the job from the Client and partition the job into multiple tasks (each task handle exact one input file) which can be concurrently processed by Compute Node. We assume Server will only process one job at a time (with many concurrent running tasks). For each input file submitted by the Client, a new thread will be initiated by the Server to process it. Each handler thread will call RPC handleMap() defined on the Compute Node side to do the Map task. For the load balancing policy, the sent task might be rejected by the Compute Node due to overloading. In this case, Server will detect the rejection and select another Compute Node to re-send the task. After all map tasks have been done and all intermediate files are generated, the sort (handleSort()) task will be initiated. We use synchronization mechanism to make sure the sort task happens after all the map tasks. So in general, Server is used to schedule, manage and monitor all the concurrent running tasks that are sent to the Compute Node.

3) Compute Node: Compute Node is used to handle the specific tasks like counting the words, calculating the sentiment score and sorting the files based on the score. There are 2 RPC defined on the Compute Node side: handleMap() and handleSort(). HandlerMap() can be called concurrently by the Server on one Compute Node to handle different map tasks at the same time. It will count the negative and positive words of one input file and store the result into the intermediate file. The handleSort() will take all the intermediate files as the inputs to do the sorting according to the score and store the results into the final file.

Note:
All components can be deployed on separate machines communicating with Thrift functions. Since we are using shared directory via NSF on lab machines, we can suppose that the directory structures and file system can be accessed by distributed machines.

Scheduling policy:

When each Compute Node is started, a particular load-probability value will be passed to the Node. We use this load value to simulate the computing load on the machine. High load will have higher possibility to inject time delay (let threads sleep for a while as if they are waiting because of the high load). We defined 2 types of scheduling policies:

1) Random

Tasks will be randomly sent to one of the Compute Nodes.

2) Load-balancing

Tasks will also be randomly sent to one of the Compute Nodes, but the Compute Node might reject the tasks based on the load-probability given to the Node. In this case, Node with high load will tend to accept less tasks.