User Document

Running each component of the system:

PLEASE ENSURE THAT YOU HAVE PLACED THE CODE IN A FOLDER WHERE YOU HAVE WRITE ACCESS, since input, intermediate and output files will be read and written from the file system.

Step 1: Compile the system using the compile script provided to generate all the .class files

```
./compile
```

Step 2: Run the job server in the first terminal window, giving its port number, chunk size (in bytes) and number of files to merge in each merge task as command line arguments.

```
java -cp ".:./libthrift-0.9.3.jar:./slf4j-api-1.7.14.jar" JobServer <port number> <chunk size> <merge file list size>
```

Step 3: Start four or more compute nodes (as many as needed) on different terminals, giving their port number, IP address and port number of job server and failure probability as command line arguments.

```
java -cp ".:./libthrift-0.9.3.jar:./slf4j-api-1.7.14.jar" ComputeNodeServer <port number> <job server IP> <job server port> <failure probability>
```

Step 4: Start the client using the following command, giving the IP address and port number of the job server as command line arguments.

```
java -cp ".:./libthrift-0.9.3.jar:./slf4j-api-1.7.14.jar" Client <job server IP> <job server port>
```

At this point, *the system has been started* and you get a UI on the client asking you to input the name of the file to be sorted.

In the next section, the messages printed at each component of the system are explained.

Details seen at various components:

1) **Job server:** Once the job server receives a file name, it divides the file into chunks and sends sort and merge tasks as explained in the design document. A task sent message is seen for each task and a task completed message can be seen for every task, both messages indicating which node the task was sent to and which node completed the task. A message is seen if a task is reassigned to another node depending on whether the original node at which the task was sent failed. Finally, we can see a job completed notification with the total time taken to execute the job, and a summary about how many compute nodes initially started the execution and how many failed when the job was running.

2) **Compute Node:** Each compute node prints out the run time for each task that it receives after completion. If a compute node survives till the job ends (it does not fail), then it also prints out the total number of tasks that it executed.

3) **Client:** The client has a UI which asks the user for the filename to be sorted. **THIS FILE NEEDS TO BE PRESENT IN THE INPUT FOLDER INSIDE THE SOURCE CODE FOLDER FOR IT TO BE FOUND BY THE SYSTEM**. Once the user enters the filename of such a file, it is sent to the job server for sorting and once the job completes, the client prints out the filename of the output file, which will be present in the output folder inside the source code folder.