# CS425 MP4 Report Wenhao Su (wenhaos3) Yichen Yang (yy18) Group 02

## MapleJuice System Design:

The purpose of this project is to build a new parallel cloud computing framework called MapleJuice, which bears similarities to MapReduce. The client can now submit maple (map) and juice (reduce) jobs to the server, and let server run the maple and juice jobs distributively on other servers, based on the distributed membership list and distributed file system implemented in the past MPs. Additionally, client can submit several maple and juice tasks simultaneously, but all those jobs are down sequentially in this distributed system.

The first phase, Maple, is invokable from the command line as:

```
maple <maple_exe> <num_maples>
<sdfs_intermediate_filename_prefix> <sdfs_src_directory>
```

The first parameter maple_exe is a user-specified executable that takes as input one file and outputs a series of (key, value) pairs. maple_exe is the file name on the sdfs. The last series of parameters (sdfs_src_directory) specifies the location of the input files. sdfs_intermediate_filename_prefix is the prefix names of the intermediate output files of maple phase. num_maples denote the number of workers that will do the maple job.

Upon receive the maple command from client, the master node will first select the worker nodes to assign the maple query. (If the number of nodes is less then num_maples, use all nodes to run the job). For each node, we assign a specific part of input files to it based on hash-based partition or range-based partition. We also assign a mission number to it. We then maintain a struct for each divided maple mission called MapleMission, containing mission_id, phase_id and a string vector of source files.

The master node will then encode the MapleMission class to a string and send to the target worker. Upon the worker receives the maple query from master, it will decode the string to find the original MapleMission class. Then it will start executing the maple job.

After receiving the maple query, the worker node will decode the query, and obtain the required files, namely the maple_exe file and all input source files from sdfs. After that, it will send back an ack to master, saying maple_mission_receive. After receiving this ack, master node will update the Stage of this mission to PHASE_II.

The worker node will then begin to execute the maple job using system command. Here we process 10 lines from the source file at a time: we read the source file, extract 10 lines and pass it to the maple_exe. We then dump the output of maple_exe to a local file called result. After all input source files are processed, the worker node will read the result file line by line and split it to at most 10 files, with the name of each file to be sdfs_intermediate_filename_prefix followed by the hashed value of the key. When the split is done, it will send back an ack to master, saying maple_mission_finished. After receiving this ack, master node will update the Stage of this mission to PHASE_III.

Then the worker node will begin to upload all the intermediate files to sdfs. When all files are uploaded to the sdfs, it will send back an ack to master, saying maple_result_uploaded. After receiving this ack, master node will update the Stage of this mission to PHASE_IV.

When all workers finish their missions, the master node will mark this maple job as finished and send back to the client that this maple job is done.

The second phase, Juice, is invokable from the command line as:

```
juice <juice_exe> <num_juices>
<sdfs_intermediate_filename_prefix> <sdfs_dest_filename> delete_input={0,1}
```

The first parameter juice_exe is a user-specified executable that takes as input multiple (key, value) input lines, processes groups of (key, any_values) input lines together (sharing the same key, just like Reduce), and outputs (key, value) pairs. juice_exe is the file on the sdfs. The second parameter num_juices specifies the number of Juice tasks. sdfs_intermediate_filename_prefix is the prefix names of the intermediate output files of the previous maple phase.

Upon receive the juice command from client, the master node will first select the worker nodes to assign the juice query. (If the number of nodes is less then num_juices, use all nodes to run the job). For each node, we assign a specific part of input files to it based on mission number of the previous maple output files. We also assign a mission number to it. We then maintain a struct for each divided juice mission called JuiceMission, containing mission_id, phase_id and a string vector of source files.

After receiving the juice query, the worker node will decode the query, and obtain the required files, namely the juice_exe file and all input source files from sdfs. After that, it will send back an ack to master, saying juice_mission_receive. After receiving this ack, master node will update the Stage of this mission to PHASE_II.

The worker node will then begin to execute the jucie job using system command. The whole process is similar to maple, while the output is a whole file that does not need to split. When the job is done, it will send back an ack to master, saying jucie_mission_finished. After receiving this ack, master node will update the Stage of this mission to PHASE_III.

Then the worker node will begin to upload its intermediate file to sdfs. When all files are uploaded to the sdfs, it will send back an ack to master, saying juice_result_uploaded. After receiving this ack, master node will update the Stage of this mission to PHASE_IV.

When all workers finish their missions, the master node will mark this juice job as finished and send back to the client that this juice job is done.

We also does error handling to MapleJuice. For each maple and juice mission, we maintain a query of free workers. When a worker is crashed, the socket from this worker to the master will be disconnected, meaning that the return value of recv function will be 0. When the master node monitored a 0 in recv, it will throw an error. In the catch stage, the master node will keep checking whether the free worker queue is not empty. If there is a free worker, then the master will send the mission structure to that worker. Unless all the workers are died, there will always be a worker doing the redistributed mission, and the task will finally be done.

**The use of MP1:**

We need to use log files to debug the whole system since we cannot use IDE or other debuggers to debug it. What we can do is to write down different logs to see which part goes wrong. Therefore, it is very useful to use MP1 to get the log files and read it in one machine to see where goes wrong.

**Measurements :**

**i. WordCount time cost for Hadoop and maplejuice with 10, 6, 3 VMs:**

For this part, we use the source of 160MB file which contains news data.
We run the wordcount on 10 VMs, 6 VMs and 3 VMs. Here is the result of wordcount:

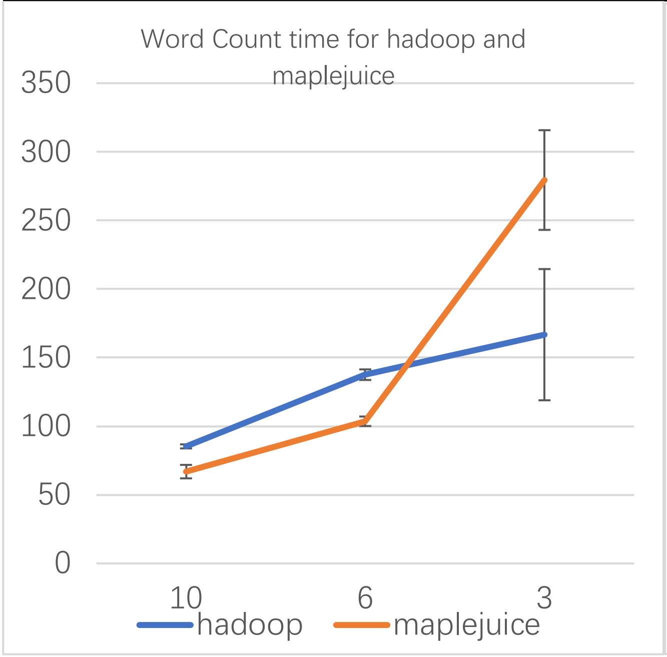
| Word Count | 10 | 6 | 3 |
|---|---|---|---|
| Hadoop (second) | 85.3225 | 137.583333 | 166.743333 |
| standard deviation | 1.485 | 3.85933068 | 47.8203684 |
| maplejuice (second) | 66.9433333 | 103.636667 | 279.433333 |
| standard deviation | 4.90858771 | 3.46214288 | 36.3222168 |

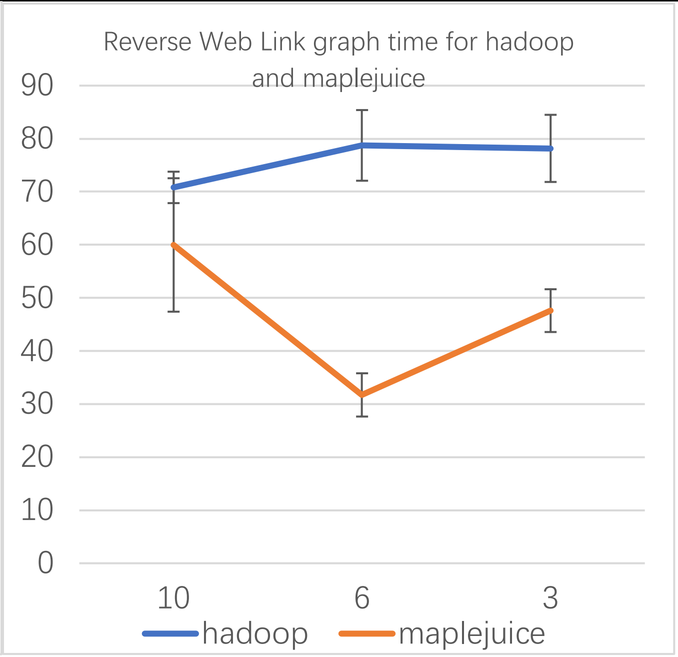
**ii. For a directed input graph, output the Reverse Web Link graph, here is the time cost for Hadoop and maplejuice with 10, 6, 3 VMs:1) file with size of 25 MB:**

For this part, we use the source of 44MB files of ego-Twitter from Stanford Large Network Dataset Collection.

We run the wordcount on 10 VMs, 6 VMs and 3 VMs. Here is the result of wordcount:

| Reverse Web link graph | 10 | 6 | 3 |
|---|---|---|---|
| Hadoop (second) | 70.8233333 | 78.7366667 | 78.1733333 |
| standard deviation | 2.95934677 | 6.65808781 | 6.32530105 |
| maplejuice (second) | 59.9766667 | 31.72 | 47.6033333 |
| standard deviation | 12.5711986 | 4.0754877 | 4.03056241 |





**Discussion for Measurement:**

From the previous data and plot, we can see that the average time for maple juice is lower than Hadoop except running with 10 VMS on WordCount. This shows that our maple juice system is very fast and efficient.

For the word count, with less VM workers, the speed will be slower. This follows our expectation since with less workers, every worker will need to process more files especially for maple. This will take longer time for one VM to process 4 maple input files than 1 file. With 10 and 6 VMs running, we can see that maple juice will take less time than Hadoop and this shows that maple juice is efficient. However, for the condition with only 2 VMs, maple juice is much slower than Hadoop. After navigating the system, we think the reason may due to the fact that maple juice can only deal with one file at a time but Hadoop may be able to parallelly deal with multiple files. When there are 5 or 9 VMs workers, since every VM only need to deal with 1-2 files, the difference is not apparent. However, when there are only 2 workers, with 5 - 8 files on each machine, it will take much longer time for linear than parallel.

For the Reverse Web link graph, we can see that in all three conditions, maple juice is faster than Hadoop which shows that our system is efficient. However, for maple juice, when there are 6 VMs, it works most fast. The reason should due to that for Reverse Web link graph, since it takes less time for processing file in maple and juice than Wordcount, the bottleneck for 10 VMs is on file transfer and stream IO. Since for 10 VMs, each worker will need to fetch 9 files and do more stream IO than 6 VMs it will take longer time. However, since the most processing file number is 2 and 3, the time should be similar. Therefore, this leads to the condition that maple juice perform best on 6 VMs.