Gohar Irfan Chaudhry (gic2) && Yang Liu (liu310)

CS425 MP4 Report

System Design:

MapleJuice is built on top of SDFS. We have built a module for MapleJucie that spawns when SDFS starts on each of the workers and it starts waiting for MapleJuice tasks.

Nodes:

There is *one master* node which is *assumed to be failure-resistant* and *N number of workers*, any of which *can tolerate failures without affecting the correctness* of the job.

Master:

When a client connects to the MapleJuice master, it will give the type of job (Maple or Juice) and the input/output details along with the number of workers it would like to use. MapleJuice master will shard the input into chunks (called *MJTasks* in our system, which could be of two types - either *MJTaskMaple* or *MJTaskJuice*, both inheriting from the *parent MJTask class*) and are put into a queue (called *MJTaskQueue*) on the master. There is also a queue of workers (called *MJWorkerQueue*) and whenever there is a free worker, it is put into the queue.

Scheduler:

There is a scheduler that is constantly running in the background (called *MJAssigner*) and it will pick a task from the queue and assign it to a free worker. Another thread called *MJChecker* runs in the background and periodically checks if the assigned tasks and their corresponding workers are still working on the task - if the worker has died then the task is put back into the queue for some other worker to consume or if the task has taken too long (timed out) then the worker is asked to stop working on it and again the task is put back into the queue.
After the Maple phase (when all Maple Tasks are done) we mark the job as done. Then the Juicer phase can be started by the client and the MapleJuice master will partition the keys and send them to the Juicers.

MapleJuice Performance Analysis:

Application:

The application run here was *WordCount*.

Input:

We used a subset from the Gutenberg Books Dataset of 100MB. The same input was used for each of the runs.

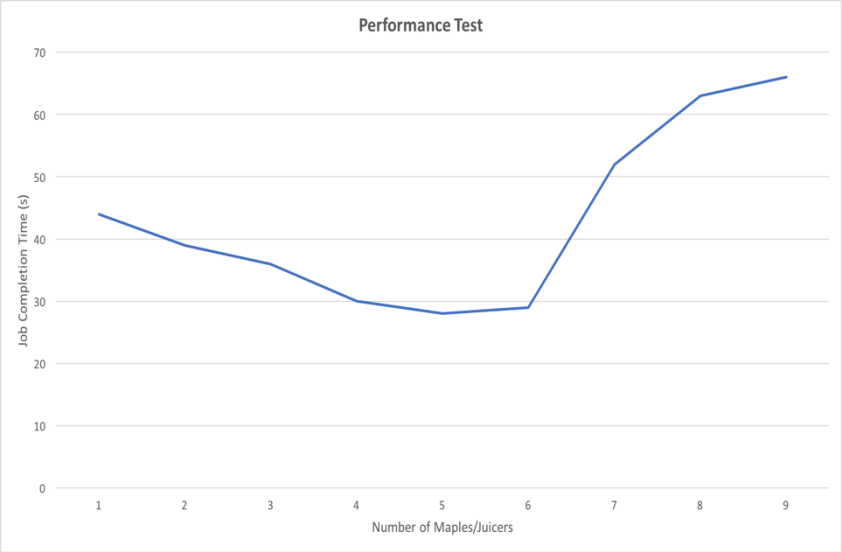
Wokers:

We ran our MapleJuice system with a different number of workers in the system, ranging from *1 worker up to 9 workers*. Each time, the number of Maples (workers running Maple) and Juicers (workers running Juice) were *kept equal* to each other.

Gohar Irfan Chaudhry (gic2) && Yang Liu (liu310)

Results:

Nodes	Average Time(s)	Standard Deviation
1	44	2.68
2	38	4.78
3	36	3.05
4	30	3.24
5	28	2.08
6	29	3.21
7	52	6.02
8	63	4.51
9	66	6.65



What do the results show?

The trend seen in the plot above shows that for an input size of 100MB, the overhead is a bit high therefore *going above 6 workers*, the *job completion time actually starts to increase* and overtakes the job completion time of 1 worker when we approach 7 workers. This is because the time taken by the job to be distributed to all workers gets high and the time taken to compile results from a higher number of pieces is more. We believe that on a much greater input, the system will perform much better with a higher number of workers.

MapleJuice Comparison with Hadoop:

Results:

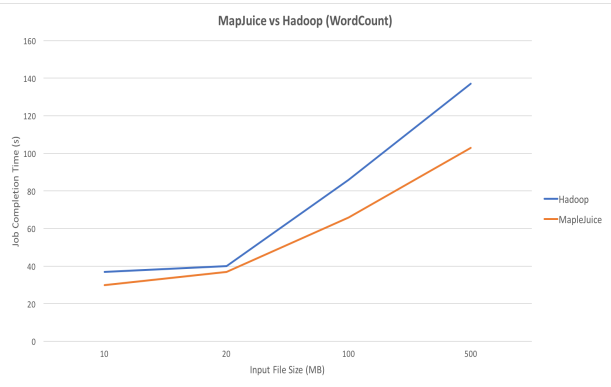
-Wordcount:

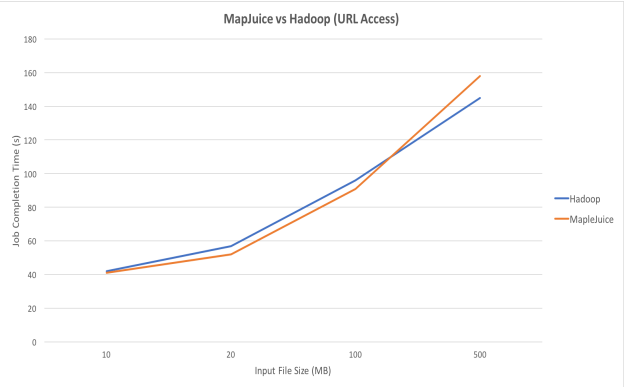
Input size(MB)	Avg of MapleJuice	SD	Avg of Hadoop	SD
10	30s	2.18	37s	1.52
20	37s	2.93	40s	2.48
100	66s	6.65	86s	7.24
500	103s	14.97	137s	13.67

-Percentage of URL Access:

Input size(MB)	Avg of MapleJuice	SD	Avg of Hadoop	SD
10	41s	2.05	42s	1.48
20	52s	1.94	57s	2.33
100	91s	5.58	96s	8.64
500	158s	16.23	145s	12.06

Gohar Irfan Chaudhry (gic2) && Yang Liu (liu310)





What do the results show?

We ran a comparison of our WordCount and URL Access Percentage applications run on MapleJuice and on Hadoop. In the WordCount case, we were outperforming Hadoop on all input sizes we used, ranging from 10MB to 500MB. In the case of URL Access Percentage application, we did well on smaller inputs of approximately 100MB but then Hadoop was able to optimize better and outperformed MapleJuice.

Input:

For WordCcount we used a subset from the Gutenberg Books Dataset and the same input (split into chunks of 5MB) was used as the input directory to MapleJuice and Hadoop, both.
For URL Access Percentage we used a subset from the Web Caching Dataset and the same input (split into chunks of 10MB) was used.

File System Configuration:

It is important to note here that we set the replication factor on the underlying distributed file systems on both frameworks (SDFS and HDFS) equal to 3. Therefore, we believe that we had set up a fair benchmarking environment.

Workers/Containers:

The number of workers used by MapleJuice were kept constant through all runs, equal to 9. However, Hadoop set the number of “containers” (similar to workers on MapleJuice) depending on the split of the input file so it ranged from 2 to 20 containers through the runs.