CS 425 - MP4 - MapleJuice

Contributors: Shubham Singhal & Nirupam K N (ss77, nirupam2) @illinois.edu

1. Design:

In this MP, we are building MapleJuice framework similar to map-reduce framework. MapleJuice framework consists of 1) Maple phase(Map Phase), 2) Juice phase(Reduce Phase), 3) Failure detection algorithm to handle the node failures and reschedule the jobs and 4) SDFS manager used to store the input, intermediate and output files.

Maple Phase: Large number of input files are partitioned based on hash and it is equally spread across all the nodes to load balance the task on each maple node. We only pass the file names to the maple nodes(Not the entire chunk of files), maple nodes copy only those files which are not present locally in their file system. Once all the required files are copied into their local file system - Maple Phase starts all the tasks and generates the intermediate output files based on key.

Juice Phase: Juice Phase receives the intermediate output received from the Maple and further reduces the intermediate results from the Maple to the final desired output that we are expecting for.

Failure Detection Algorithm: Whenever a particular node fails, all the file names that are used for the Maple or Juice task will be gossiped to master node which is currently up and running to reschedule the task as we have already maintained the list of files used for that node in a dictionary.

SDFS manager: Input files are put into sdfs directory which are to be used by Maple. It also keeps track of all the intermediate output files generated by Maple Phase based on the Key that needs to be sent to Juice Phase for further computation. And finally the reduced output file is also stored in the sdfs directory.

2) Chosen Application:

2A) Word Count Application:

This is a simple word count application, which gives the total count of a word present in all the input files. We checked the performance of wordcount application on both Hadoop MapReduce framework and on our Maple Juice framework as well.

Table 1

	Word Count(3M/3J)	Word Count(6M/6J)	Word Count(9M/9J)
R1	75.0542	63.9952	57.9672
R2	66.3540	62.0782	59.4947
R3	70.6543	61.9907	56.8754
Average	70.6875	62.6880	58.1124
Std Deviation	4.3502	1.1329	1.3157

Y-Axis - Time in seconds; X-axis: Readings ; M-Maple; J-Juice

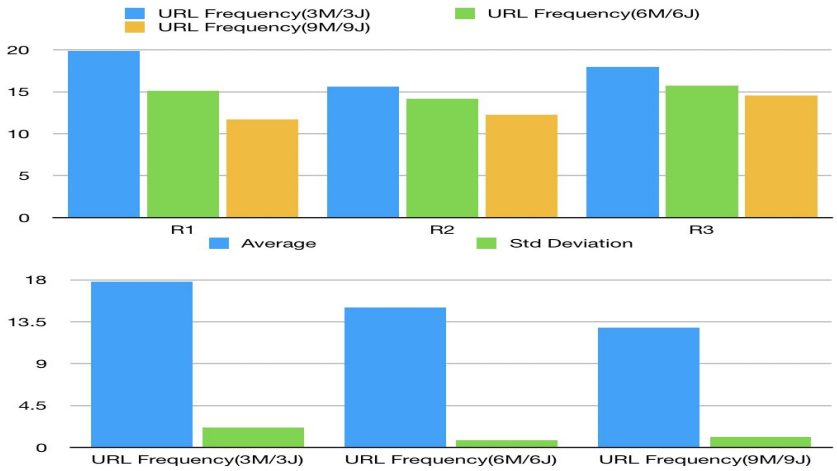
2B) URL Frequency Application:

For a given web proxy log, we process the data and output the percentage of access that go to that particular URL. We checked the performance of URL Frequency application on both Hadoop MapReduce framework and on our Maple Juice framework as well.

Y-Axis - Time in seconds; X-axis: Readings ; M-Maple; J-Juice

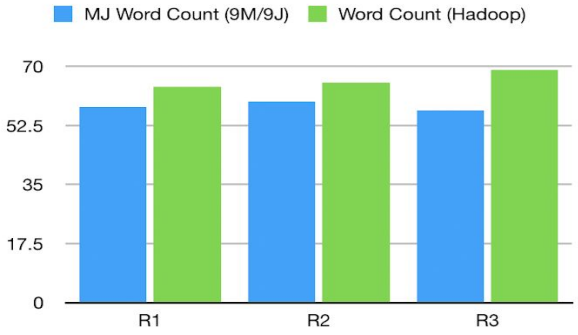
Table 1

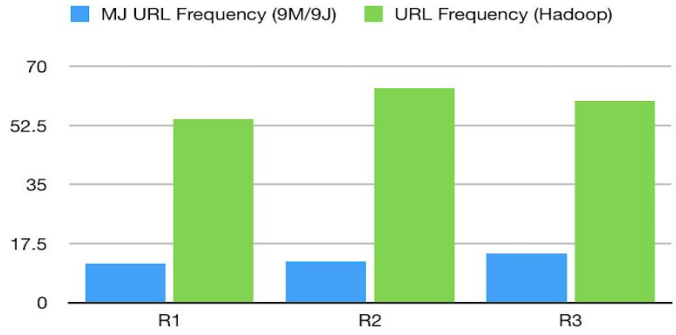
	URL Frequency(3M/3J)	URL Frequency(6M/6J)	URL Frequency(9M/9J)
R1	19.8659	15.1349	11.7105
R2	15.6247	14.1783	12.2871
R3	17.9466	15.7356	14.5739
Average	17.8124	15.0163	12.8572
Std Deviation	2.1239	0.7854	1.1544

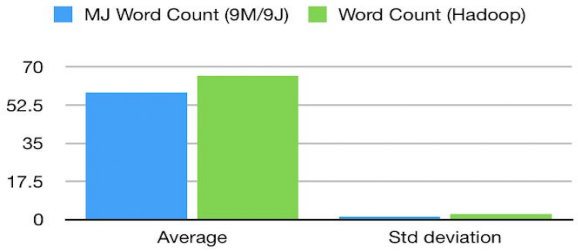


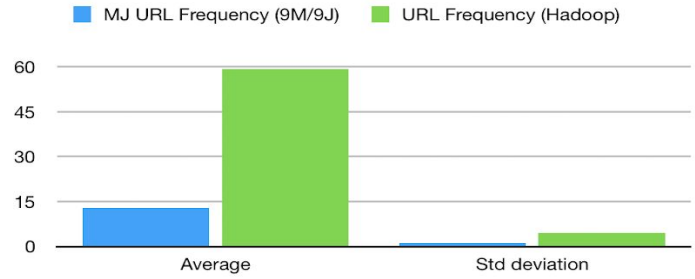
Maple-Juice Vs Hadoop

	MJ Word Count (9M/9J)	Word Count (Hadoop)	MJ URL Frequency (9M/9J)	URL Frequency (Hadoop)
R1	57.9672	63.9507	11.7105	54.3845
R2	59.4947	65.138	12.2871	63.6209
R3	56.8754	68.8932	14.5739	59.7649
Average	58.1124	65.9939	12.8572	59.2568
Std deviation	1.3157	2.5801	1.1544	4.6391









Maple-Juice and Hadoop Comparison

Y-Axis - Time in seconds; X-axis: Readings ; M-Maple; J-Juice

In both applications, Maple-Juice is performing better than Hadoop. For experimentation purposes, we have taken around 100 keys with each key's value around 30-40K. As we increase the number of keys, Maple-Juice performance will not be better than Hadoop because we are storing the Maple's output as intermediate output file which is per key based in the SDFS. In our design of SDFS file system, we have a data replication logic where each file will be saved on 4 nodes as a replica. As the number of keys increases, replication overhead increases, thereby decreasing the actual performance of Maple-Juice compared to Hadoop. Thus for large number of keys, hadoop's performance will outperform Maple-Juice. In the URL frequency application, the execution time is less because we have used only one key chained with multiple URLs and their count. As a result, only one intermediate file is put into SDFS. This significantly reduced the execution time as compared to hadoop.