1 Design

MapleJuice is a batch processing system that works like MapReduce.

1.1 Programming Model

MapleJuice consists of two phases of computation – Maple (brother of Map) and Juice (sister of Reduce). Maple takes a set of data and transforms their individual lines into a list of key-value pairs. Juice combines the output from Maple into a smaller list of key-value pairs. However, the Maple function processes 10 input lines from a file simultaneously at a time, while the traditional Map function processed only one input line at a time. The Maple and Juice tasks are user-defined. Users can upload any Maple/Juice executable files that fit the programming model to the system. For simplicity, our system can only run one phase at a time.

1.2 System Architecture

MapleJuice is composed of two major components: i) single master node for task scheduling and coordination, ii) worker nodes for task processing. Each master/worker also provides a client interface for interacting with users. It uses our Distributed Group Membership Service in MP2 for failure detection and Simple Distributed File System (SDFS) in MP3 for storing the input data and the results of the MapleJuice jobs.

Workflow: Users submit jobs on clients, which then send job requests to the master node. When doing a Maple job, the master node partitions the input data and evenly distributes the partitioned data to a set of selected workers. Each worker processes its share of data by repeatedly calling the user-uploaded Maple executable. The generated key-value pairs are sent back to the master to be gathered and written to SDFS, one file per key. When doing a subsequent Juice job, the master node shuffles the keys and allocates them to another set of selected workers. The workers process their dispatched task by repeatedly calling the user-uploaded Juice executable. The processed results are sent back to the master to be combined and written to SDFS.

Scheduling: The master node is in charge of all the scheduling work. It maintains first-in, first-out job queue, allowing at most one job to run at a time with all subsequent jobs waiting in queue. When dispatching tasks to workers, it prefers worker nodes on which replicas of input files already exist to reduce file I/O. When doing failure recovery, it favors free workers (if any) to restart the task previously running on the failed worker.

Fault Tolerance: MapleJuice can tolerate up to 3 simultaneous worker failures, limited by the replication factor (4) used in SDFS. A master failure cannot be tolerated though. On the master node, a worker-task mapping table is maintained. When a working worker is reported as failed, the master node retrieves the information of its running task(s), and selects another worker (free worker preferred) to restart the task(s).

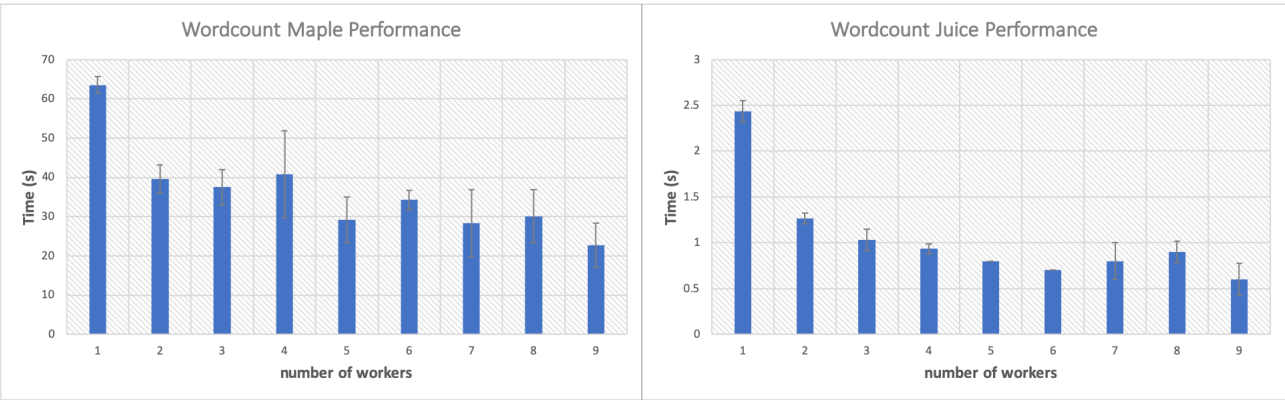
2 Comparison against Hadoop

We choose two simple applications to compare the performance of our MapleJuice with Apache Hadoop MapReduce:

1. Wordcount,
2. For a web proxy log, output for each URL what percentage of access go to that URL.

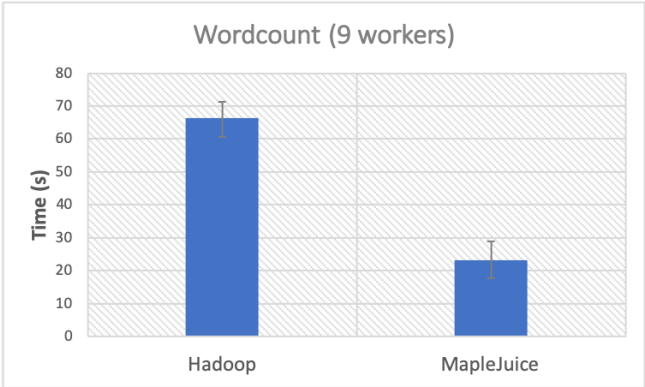
2.1 Wordcount

For the experiments, we use 9 files with roughly equal size. Each file has around 4400 to 4500 lines. Each line consists of 300 to 500 words. Range partitioning is used for shuffle grouping in MapleJuice. The entire dataset is around 110 MB. Each column summarizes the result of 3 experiments under identical settings with x number of workers. For the performance of the distributed system, we simply focus on the completion time. The results corresponding to the performance of maple phase and juice phase are shown in the following graph.



As we can observe, as the number of workers increases, there is a decreasing trend in the time used to finish both maple phase and juice phase. Using 9 workers (the maximum number of workers) can improve the performance of one single worker by 64% during maple phase, and 75% during juice phase. The comparison between the performance of Hadoop and MapleJuice is shown in the following graph.

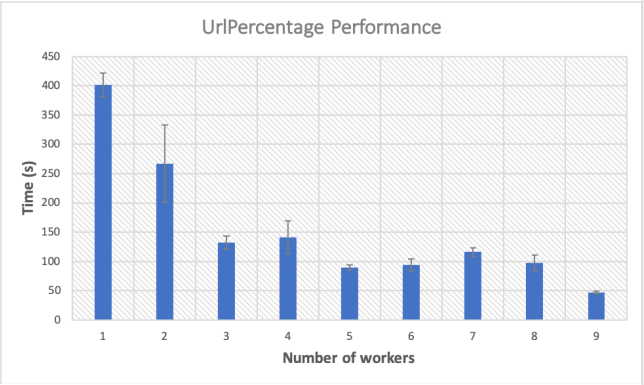
We also notice that MapleJuice's performance is not monotonically decreasing when the number of workers increases. This is presumably because of the coarse data partitioning that we apply. For example, in the 4 workers case, there will be 1 worker that receives 3 files and the other 3 workers receive 2 files. Then the performance of maple will be determined by the worker with 3 files because he will take the longest. If the worker who handles 3 files happens to be a worker with slower CPU or the network is in bad condition, then there will be a significant drop in the performance. The same reason also applies to why the maple performance increases by 24.4% from 8 workers to 9 workers - no one has 2 files, and every one will finish around the same time. One potential solution will be a more fine-grained partition that splits each file into equal-sized pieces, so that the workload can be evenly distributed, and avoid actively making nodes stragglers.

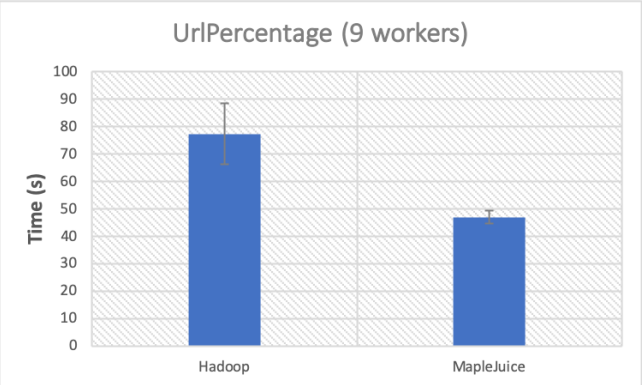


It takes MapleJuice on average 23.33 seconds to finish running the entire wordcount job on 9 workers, which is 64.8% faster than Hadoop. Hadoop is known for handling big files and it seems to have poor optimization with handling small files that are far smaller than the Apache Hadoop HDFS default block size (128MB).

2.2 Percentage of access going to each URL

For this job, we focus on MapleJuice's performance when dealing with chained jobs. We generated a dataset that has 9 files. Each file is around 10 MB, and contains around 300000 lines, where each line contains only the URL. Range partitioning is used for shuffle grouping in MapleJuice. For the first MapleJuice job, the output is (key, value) pairs of counts of each URL. This result will then be used as the input for the second MapleJuice job, where the output is (key, value) pairs of percentages of access for each URL. The average results of the chained MapleJuice jobs regarding the number of workers is shown in the graph below, along with the comparison with Hadoop.





Same as wordcount, there is a decreasing trend in completion time when the number of workers increases. MapleJuice still outperforms Hadoop in this application.

We also notice that the standard deviation of completion time can vary by a lot. For example, the time is considerably large when number of workers is 2. By using MP1, we looked into the log and found out that the local computation time remains relatively the same for each executable file on every worker node, but the time used to disseminate SDFS files is vulnerable to network condition, which we conclude as another bottleneck for MapleJuice.