MP3 Report - MapleJuice

Alvina Waseem (awaseem2@illinois.edu)
James Timotiwu (jit2@illinois.edu)
Kartikeya Sharma (ksharma@illinois.edu)

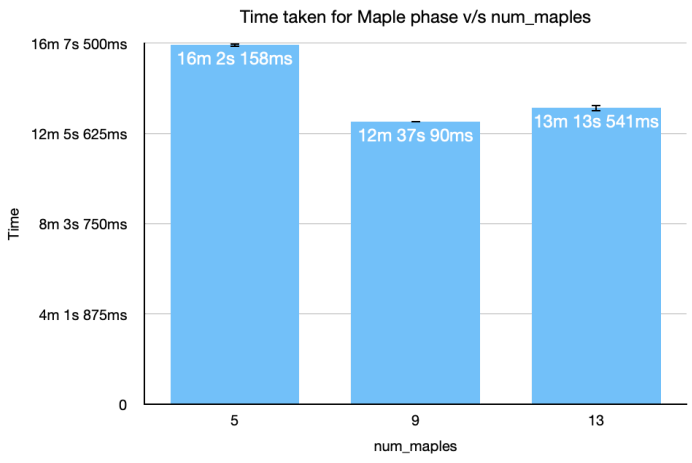
Design: We have built a Map-Reduce (Maple-Juice) framework, on top of our failure detector (mp1) and simple distributed file system (mp2). Salient Features:

1. **File Chunking**: Each file, when uploaded to SDFS, is broken down into blocks based on the size specified in the configuration file, rounded to the nearest newline.
2. **Master:** Responsible for receiving Maple and Juice requests from clients, and allocating maple and juice tasks. Also responsible for handling failures during Maple/Juice tasks.
3. **Maple Phase:** Called by any client. Request is sent to master, which creates maple tasks equal to the number of file blocks or the num_maples argument, whichever is lower. Each maple task is assigned to one of the replicas of a block randomly and processes records within it. Each worker machine handling a maple task writes all outputs for a certain key to a local file, and then notifies master upon completion of all its maple tasks.
4. **Juice Phase:** Called by any client after successful completion of a prior maple task. Each juice task is assigned keys by a partitioner (both range and hash partitioners are implemented). These are then assigned to worker VMs using hash partitioning. Each key is paired with a juice task. The worker VM performs a remote read for maple output associated with a key, merges them with shuffle sort, and runs the values on a juice application. The final outputs are collected by the master and uploaded to SDFS. The output can be retrieved from SDFS using the get command.
5. **Failure handling:** If a node running a maple task on a block fails, the maple task is assigned to another node holding a replica of that block. If a node running a juice task fails, it is also assigned to another node.
6. **API:** The framework was written in Go. An API is provided which requires the User to simply write Maple() and Juice() functions for Maple and Juice tasks.

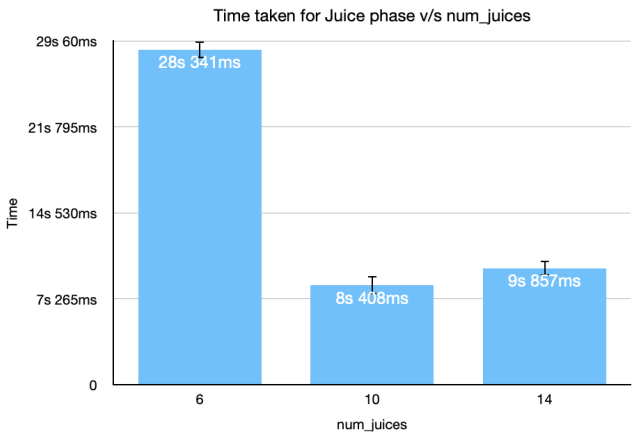
Experiments :

1. **Condorcet Voting Application:** Varying num_maples and num_juices

Maple Phase (sum of both phases)



Juice Phase (sum of both phases)



Here, we plot the time taken for both phases of the condorcet voting application (10 candidates) from HW1 against num_maples and num_juices. This was done for a block size 16 MB, for an input file of 136MB (synthetically generated dataset) which resulted in 9 blocks. Thus when num_maples = 9, we see the lowest value for time taken. It increases on either side, since for lower values, less parallelism is achieved than optimal, and for higher values some of the maple tasks do nothing and only add overhead. Similarly, for num_juices = 10, we see the best performance, since we had 10 VMs and having 10 juice tasks maximizes parallelism.

2. **Wine Reviews Application**

We plot the time taken for one phase of the wine reviews dataset application. Similarly, the block size is set to 16MB, the input file size is 50MB, resulting in 4 blocks total. Maple performance peaks at num_maples = 4, which is expected. The application counts over many keys, so 10 juice tasks across 10 running nodes provided the best performance.

Comparison Against Hadoop:

1. **Condorcet Voting:**
   HDFS block size was set to 16MB, and the number of reducers was set to 10 for fair comparison. The dataset size was 135 MB, having condorcet voting data for 10 candidates.
   MapleJuice Time: 12 min 45 s
   Hadoop Time: 24 min 3 s
   We can see that MapleJuice does better than Hadoop. This is because the dataset, despite being ~135 MB is still smaller than datasets required to get the most out of Hadoop, due to the high cost of setup and intra-block data sharing in Hadoop. Further, we ran into issues where Hadoop would allocate the maximum amount of memory for our machines, which would freeze the program occasionally. This is not an issue for MapleJuice. However, we can see that compared to even smaller datasets (wine reviews application), it does better relatively.

2. **Wine Reviews Application:**
   HDFS block size was set to 16MB - the number of reducers was set to 10. The dataset is mid-size, 50MB in all, so around 4 blocks were created for each test - corresponding, 4 maple tasks were performed. The application consists of a single map phase and a single reduce phase.
   MapleJuice Time: 7s
   Hadoop Time: 1 min 51s
   MapleJuice performs well for this dataset in particular as less time was spent on the overhead of the MapReduce engine. The application itself isn't complex and does not directly benefit from any optimizations offered in Hadoop.