CS 425 MP3 Report (g33 - James Wang & Rishi Desai)

Design

Our MapleJuice processing system involves two types of nodes: a master and some number of workers. MapleJuice uses SDFS to store task output and input files, and the MapleJuice master node is also the master for SDFS. The master node is not fault tolerant, meaning that if the master node fails, then MapleJuice will not work.

When a task is submitted to the master, the master partitions the task into smaller shards, which get distributed to the worker nodes. Our design assumes that the input corpus is evenly sharded, since the partitioning works on individual files, not looking at their contents. When a worker node finishes its task shard, it uploads the output files to SDFS and notifies the master.

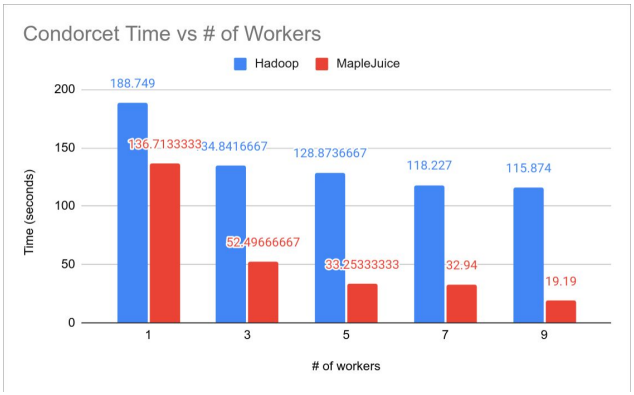
If a worker node fails, then the master will look at all task shards that the node was responsible for, then distributes them among the remaining worker nodes that have the least load.

Our solution could be faster if the master node sharded input such that each worker gets the same number of lines instead of using the file count. Additionally, we believe our usage of SDFS could be improved, since each node will query the master when retrieving SDFS files, instead of only doing so if its own SDFS server doesn't have it.

Analysis

Condorcet

Our Condorcet dataset consisted of 9 randomly generated files with 500,000 rows each.

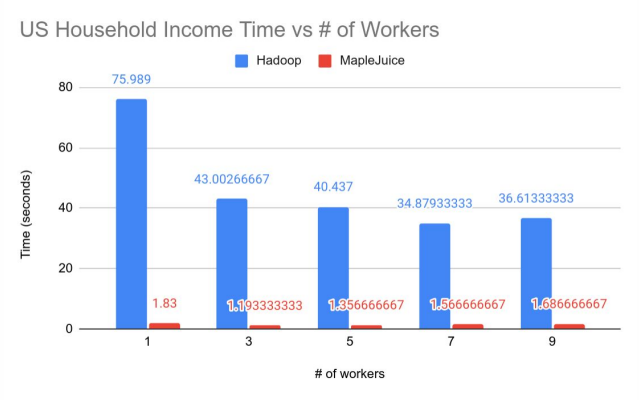


# workers	std (Hadoop)	std (MapleJuice)
1	4.133046697	0.9457448564
3	2.541544084	2.120786018
5	2.603115505	0.560743554
7	11.0187154	0.2858321186
9	10.49854376	0.1307669683

We can see that the time it takes to finish Condorcet decreases as the number of worker nodes increases. However, we see that MapleJuice is consistently faster than Hadoop. The speed difference here was unexpected, since simply running the Hadoop map/reduce scripts locally resulted in a significantly faster runtime. We attribute Hadoop's slowness to the limited resources available on the VMs (1 CPU and 2gb of RAM), overhead of the JVM, and the fact that Hadoop is optimized for much larger and more complex workloads than the ones we tested with. When measuring the resources used when running MapleJuice compared to Hadoop, we observed significantly less memory usage.

US Household Income

We used a dataset from the US census bureau on every household in America. This dataset was 30.9 MB and we created nine 3.3 MB shards. Then we wrote a MapReduce program that finds all of the households in the US from the midwest and sums up the number of households in each income range (income ranges defined here).



# workers	std (Hadoop)	std (MapleJuice)
1	2.52654052	0.05567764363
3	0.8488818135	0.03214550254
5	5.568073365	0.1096965511
7	2.654138718	0.1193035345
9	4.654104031	0.2040424792

With this dataset we continue to see MapleJuice outperform Hadoop. We can attribute this to the overhead with Hadoop on the resource constrained VMs. MapleJuice has similar times as the number of workers grows since the shard sizes aren't that big. The Hadoop times saturating when there are more than 2 workers is also connected to the shard sizes not being too large.