CS 425 MP4 Report: MapleJuice

MapleJuice is a distributed computing framework designed for the efficient execution of Maple (Map) and Juice (Reduce) jobs in a parallel and fault-tolerant manner. The system operates on a leader-worker architecture, where the leader orchestrates job scheduling and execution. New Leaders are elected using the Bully Election Algorithm.

The leader schedules Maple and Juice jobs in FIFO order by default. Maple and Juice jobs are executed on worker nodes. By default all nodes in the system are worker nodes including the leader. When a worker receives a Maple or Juice Job a corresponding Maple or Juice Phase is entered.

Maple Phase

1. Leader fetches all input file names from the SDFS (created in MP3) and assigns partitions of the file names (hash-based or range-based) to other nodes based on the membership list (maintained by Gossip Failure Detection from MP 2).
2. Leader evenly spreads out num_maples across all worker nodes in the membership list, assigning each worker num_tasks.
3. Leader waits for all Map Tasks to finish before proceeding, re-assigning failed Maple Tasks to other unassigned workers.
4. Leader receives all keys from Map Tasks, fetches all files from the worker nodes to aggregate them resulting in multiple intermediate files, each per key which are uploaded to the SDFS.

Juice Phase

1. Leader fetches all intermediate key files from the SDFS and assigns partitions of those keys (hash-based or range-based) to worker nodes based on the membership list.
2. Leader evenly spreads out num_juices across all worker nodes in the membership list, assigning each worker num_tasks.
3. Leader waits for all Juice Tasks to finish before proceeding, re-assigning failed Juice Tasks to other unassigned workers.
4. Leader receives all keys from Juice Tasks, fetches all files from the worker nodes to aggregate them resulting in a single output file which is uploaded to the SDFS.
5. Optionally deletes all intermediate files from the Maple Phase.

Overall, workers are designed to: (1) Receive Maple Task or Juice Task from Leader. (2) Fetch the necessary input files from SDFS. (3) Spawn num_tasks threads and execute the maple_exe or juice_exe as a subprocess. (4) Aggregate the output from the maple_exe or juice_exe.

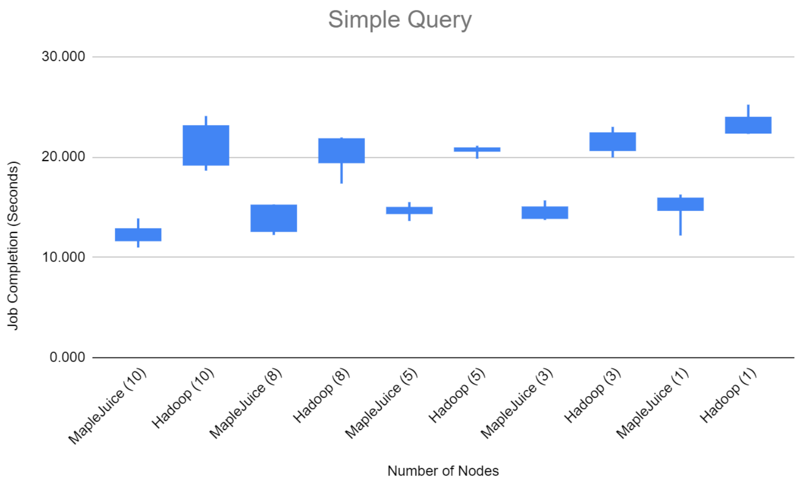
All messages between the leader and workers are through RPCs (for Python we used the RPyC module).

MapleJuice also supports SQL queries that are parsed using Python SQL libraries and translated to corresponding RPCs that schedule the appropriate Maple Job(s) and Reduce Job(s).

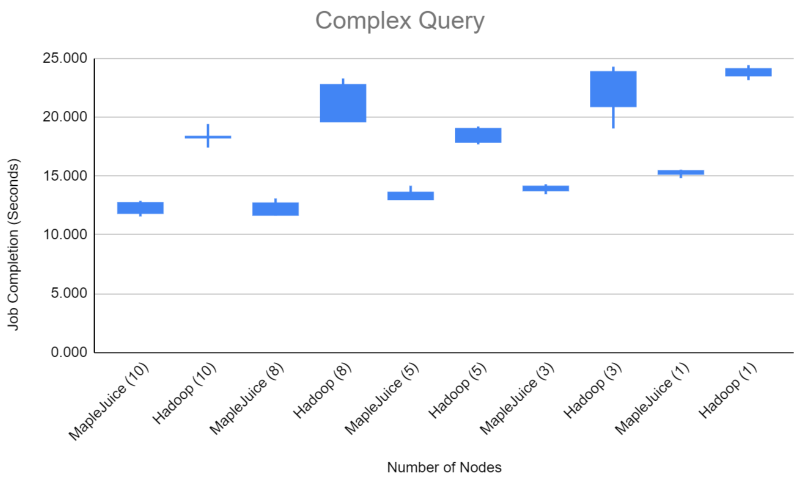
For Simple Queries we used grep keyword searches. For Complex Queries we used more complicated regex, such as searching for a specific pattern in a specific column of a comma delimited CSV.

- https://www.kaggle.com/datasets/ironicninja/icm-problem-d?select=prediction.csv (~300 MB)

(Filter 1 & 2): MapleJuice slightly outperforms Hadoop for simple and complex greps on small amounts of data.



For MapleJuice, shrinking a cluster of 10 nodes to 3 nodes results in a drastic decrease in performance. As expected, the performance drop is somewhat proportional to the number of workers combined with the overhead of the SDFS.



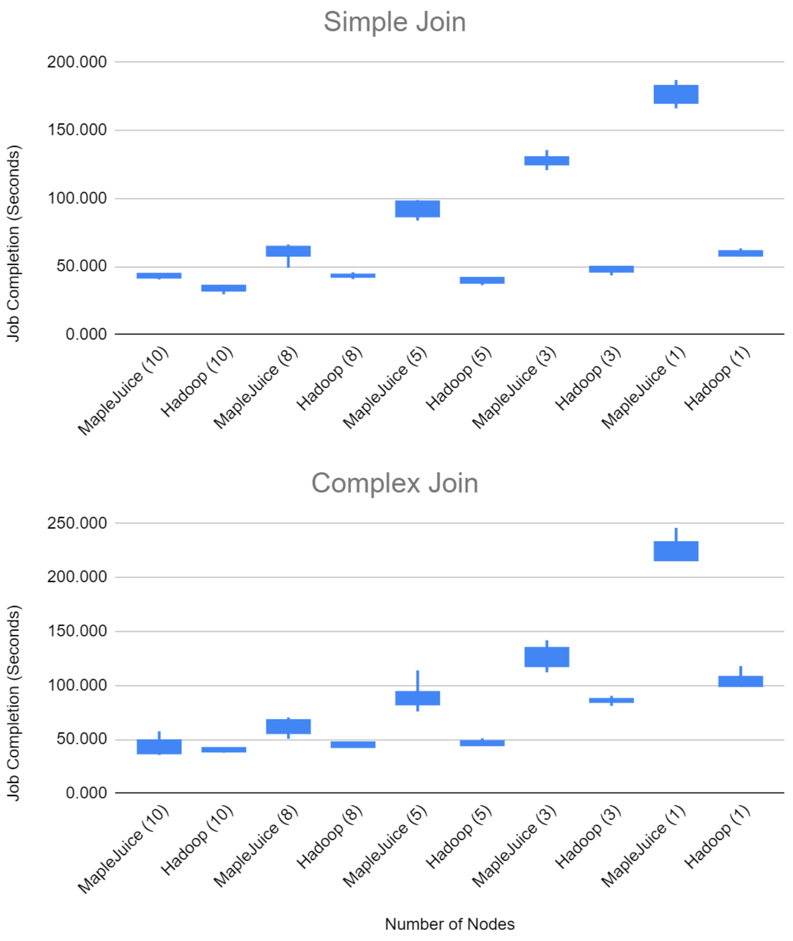
For Both MapleJuice and Hadoop, varying the grep query did not result in any meaningful performance change. This is because the number of generated intermediate files is largely unchanged.

For the next set of benchmarks, since our system is limited to SQL INNER JOINS on a single column, we define complexity by the number of keys generated. The Simple Join had 200 unique join keys while the Complex Join had ~1,000 unique join keys. Due to resource contention in MapleJuice from a high number of join keys, we had to keep the dataset size small (~200 MB) to maintain a "fair" comparison. We cross referenced the Spotify Artists with the US Baby Names (modified to meet size requirements and ensure results).

- https://www.kaggle.com/datasets/ironicninja/icm-problem-d?select=prediction.csv
- https://www.kaggle.com/datasets/kaggle/us-baby-names

(Join 1 & 2): As expected Hadoop outperforms MapleJuice for a few reasons:

(1) Hadoop can map different datasets to different mappers. Maplejuice requires 2 different Map Reduce stages to read the 2 datasets to one common dataset and a final map-reduce to join.
(2) Hadoop's sorting of keys allows for memory efficient reduce-side joins. MapleJuice's reduce-side join quickly adds to resource contention.
(3) The throttling mechanism to avoid resource contention is cluster size independent. For fewer nodes, resource underutilization results in exponentially increased computation times for.



The significantly higher times on 1 node configurations is also due to resource contention. For Hadoop, non-critical computation services, steal resources away from the critical computations. Similarly, MapleJuice's NameNode steals resources from the critical computations. This is ok since a 1 node configuration doesn't make any sense beyond dev and test environments.

A notable factor impacting the performance of MapleJuice was the number of keys generated from the Map phase. Writing a large amount of small keys results in overutilization SDFS resources (i.e. flooding the NameNode with requests) to move small amounts of data which is suboptimal. In other words, "using a sledgehammer to crack a walnut."

We tried to solve SDFS's "many small files" issue by tweaking the resources allocations for concurrent PUTs, DELETEs, GETs and Replication Factor but run into a trade off of resiliency (Replication Factor/Speed) and Job Completion Rate/Throughput (allocating more resources for concurrent GETs and PUTs and less for Replication and DELETEs).