Jiachen Yuan & Sizheng Zhang 11/27/23

CS-425 MP4 Report

Design & Algorithm Overview

The high-level design of our system is to have an elected leader server that processes all the MapleJuice requests from any client. The leader has two major responsibilities:

1. partitioning and assigning maple/juice task workloads to different worker nodes
2. monitor the progress of each worker and reschedule any failed maple/juice tasks

Maple

Upon receiving a maple request, the leader distributes tasks to workers based on input file size. The workers will receive the work assignments in the form of [{file1: linesToProcess}, {file2, linesToProcess}...]. Each worker will then fetch the required files from SDFS and execute the maple program. After finishing its work, the worker sends a response to the leader, who, after all workers have done, send a "completed" notice to the initiator client of this maple job. When detecting the failure of execution from any worker, the leader will soon assign the job of that worker to the next available worker in a round-robin manner.

Juice

For juice requests, the leader allocates tasks based on partitioning technique specified by clients (range or hash). After receiving the juice execution request from the leader, workers will fetch the juice exe file and the corresponding intermediate files with its assigned keys and run the juice exe program on each file. The notification of finishing a job and failure handling is similar to that of the maple phase.

SQL

On top of the MapleJuice system, we built another SQL layer. When the client sends a SQL request, we will parse the input and convert them into suitable maple and juice tasks.

Use of Previous MPs

We used MP3 as our underlying filesystem that supports Maple and Juice. We used MP2 as our underlying failure detector. We used MP1 to grep log and help debug the system.

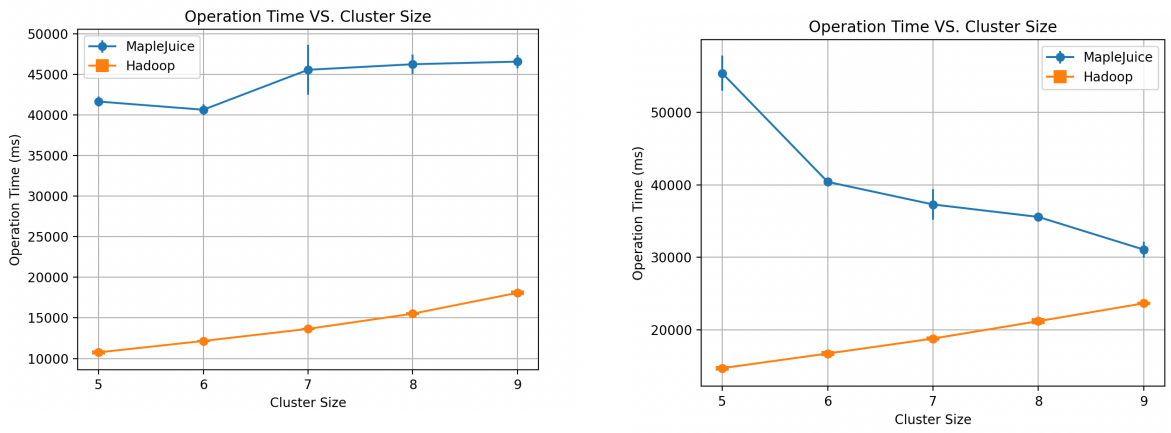

Figure 1. Operation Times for Dataset 1 Filter Task 1 (**Left**) and Filter Task 2 (**Right**)

Measurements

Dataset 1: We selected the 2020 presidential election twitter data from Kaggle. The data can be found here: https://www.kaggle.com/datasets/manchunhui/us-election-2020-tweets/data.

Filter 1: SELECT ALL FROM Dataset WHERE tweet REGEXP @realDonaldTrump

Analysis: Figure 1 (**Left)** shows that MapleJuice takes about four times longer for average execution compared to Hadoop. Nevertheless, both lines in the plot follow the same pattern: as the cluster size increases, the time for filter query operation goes up. We think this happens because the basic query doesn't take much time, and adding more machines to the cluster doesn't significantly speed up the query but adds more system overhead (like increased inter-node communication), slowing down the whole process. Therefore, as the number of nodes goes up, the performance of both MapleJuice and Hadoop decreases.

Filter 2: SELECT ALL FROM Dataset WHERE tweet REGEXP (#[A-Za-z0-9_]+).*(#[A-Za-z0-9_]+)
Analysis: Figure 1 **(Right)** shows MapleJuice performs worse than Hadoop. However, as the cluster size increases, Hadoop's performance declines (like the simple filter query), while MapleJuice's improves. This is because our queries get more complex, demanding more CPU resources. Adding more workers aids load balancing. In Hadoop, the benefit of more machines is offset by system overhead, causing a negative correlation between cluster size and SQL operation time in the Hadoop line on the graph.

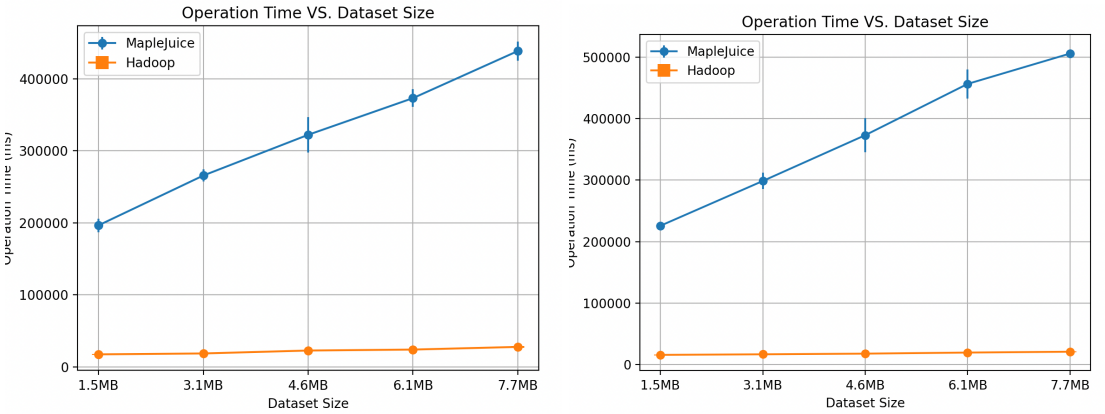

Figure 2. Operation Times for Dataset 1 Join Task 1 (**Left**) and Join Task 2 (**Right**)

In order to save time for our experiment while making sure that the runtime of our program is around tens of seconds, we truncated the dataset to be less than 10 MB to perform the two JOIN operations.
Join1: SELECT ALL FROM Dataset1, Dataset2 WHERE Dataset1.user_id = Dataset2.user_id; This join query uses the same dataset twice but with different sdfsnames (self join). We use this as the complex query because the number of matches would be large (quadratic).
Analysis: In Figure 2 (**Left**) our system's overhead exceeds that of Hadoop due to differences in scheduling efficiencies and system overheads for file distribution.

Join 2: SELECT ALL FROM Dataset1, Dataset2 WHERE Dataset1.tweet_id = Dataset2.tweet_id; This join query is simpler because the tweet IDs are unique and the number of matches is smaller (linear).
Analysis: In Figure 2 (**Right**), our system's overhead increases more rapidly due to unique IDs in datasets, resulting in at most one matching row per key. This task is less CPU-intensive and more memory-bound, given the numerous intermediate files with IDs as keys.

Dataset 2: We chose the credit card fraud detection dataset from Kaggle. The data can be found here: https://www.kaggle.com/datasets/kartik2112/fraud-detection?rvi=1. Notice the original csv file does not have an id field at the front of the schema, and we added it so the csv is easier to handle.
Filter 1: SELECT ALL FROM Dataset WHERE merchant REGEXP Herzog
Analysis: Figure 3 **(Left)** is consistent with the one from Filter 1 of Dataset 1, except that in this graph the performance of MapleJuice is only about 2 times slower than Hadoop. This might be because of the fact that Dataset 2 is inherently less complex than Dataset 1, as Dataset 1 contains a lot of long twitter texts that are difficult to run regular expressions on.

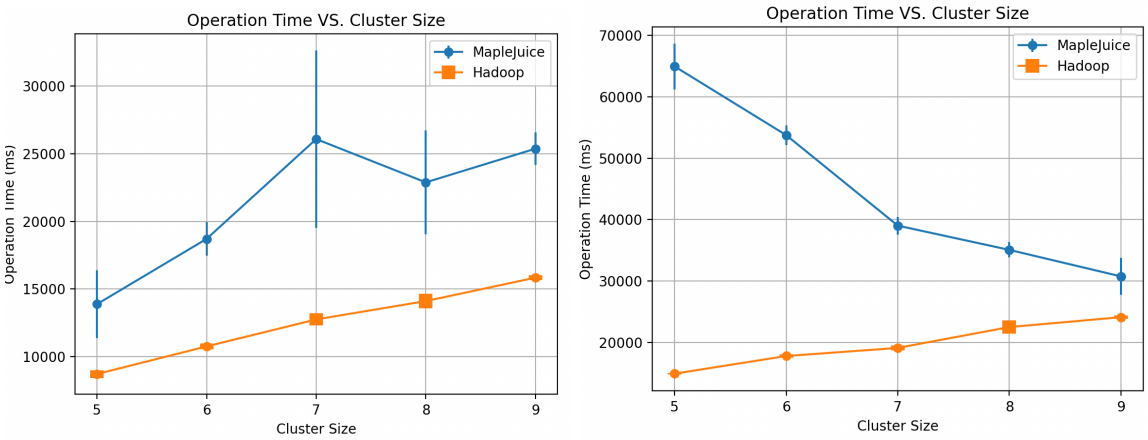

Figure 3. Operation Times for Dataset 2 Filter Task 1 (**Left**) and Filter Task 2 (**Right**)

Filter 2: SELECT ALL FROM Dataset WHERE trans_date_trans_time REGEXP 2019.* (18|19|20|21|22|23):[0-5][0-9]:[0-5][0-9];
Analysis: The above graph (**Right**) is consistent with the one from Filter 2 of Dataset 1, and we believe that the same reasons contribute to the similarity between these two experiments.

Join 1: SELECT ALL FROM Dataset1, Dataset2 WHERE Dataset1.cc_num = Dataset2.cc_num; This join query uses the same dataset twice but with different sdfsnames (self join). We consider this as a complex query because the number of matches would be large (quadratic) and the load for our Juice phase would be very large despite small input files.
Analysis: Figure 4 (**Left**) indicates that both our system and Hadoop experience longer operation times with larger input datasets. However, our system performs less effectively than Hadoop, primarily due to communication cost and scheduling decisions. Additionally, the join's characteristics, with relatively fewer intermediate keys and many matching rows, make it inherently more CPU-bound, resulting in less dramatic increases in operation times with larger file sizes in our system.

Join 2: SELECT ALL FROM Dataset1, Dataset2 WHERE Dataset1.id = Dataset2.id; We consider this query simple on fraudTrain.csv and fraudTest.csv because the load on the Juice phase would be smaller since id in each dataset should be unique (fewer matching rows).

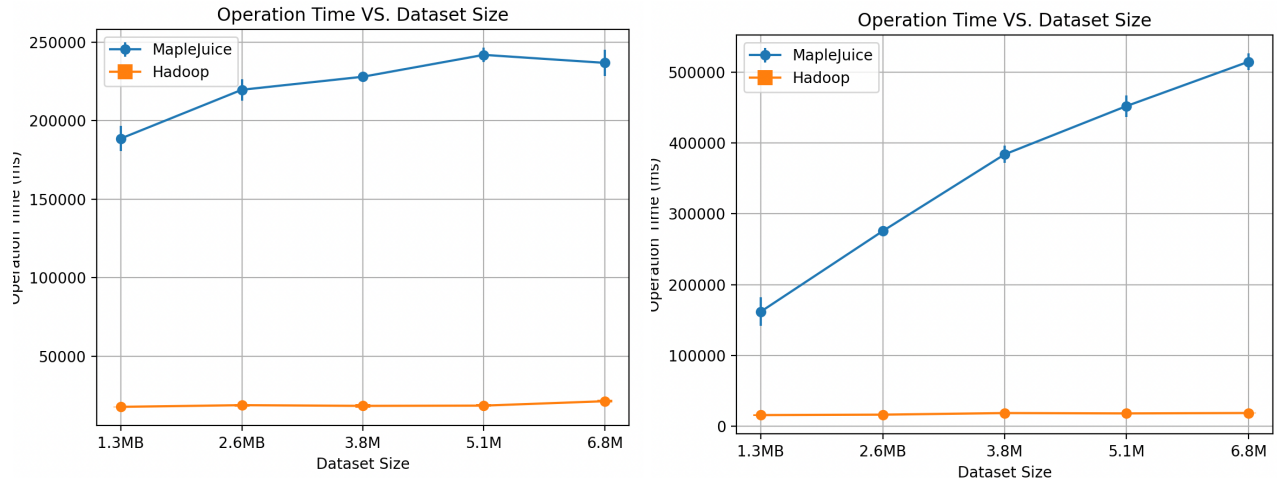

Figure 4: Operation Times for Dataset 2 Join Task 1 **(Left)** and Join Task 2 **(Right)**
Analysis: In Figure 4 (**Right**), similar trends to the previous Dataset 1 Join task 2 are observed for similar reasons.