MP4 Report

Feize Shi (feizes2) and Dongheng Lin (dl58)

Design

The basic design of our MapleJuice implementation referenced the Hadoop and MP4 specifications, in which we have a MasterScheduler that receives job execution requests and schedules them. We run an additional SDFS system we developed in MP3 to store and maintain the input files, executables, intermediate files, and final outputs. Finally, we also have the system equipped with a client, allowing users to interact.

1. MasterScheduler

The MasterScheduler is equipped with a FIFO queue that processes the incoming jobs individually; therefore, a system can process one SQL command without interference with other commands. When receiving different inputs, the scheduler will take different Maple/Juice executables to create JobTrackers. The different inputs are handled in the Client part, which parses the SQL command into different jobs.

2. JobTracker/NodeManager

JobTracker works with NodeManager to execute Maple/Juice jobs. Simply put, JobTracker (per job) is responsible for splitting the Maple/Juice job into multiple (user-specified number) fine-grained tasks, then scheduling them to NodeManager for specific execution, tracking the execution of the tasks if they fail, then re-schedule the task to another NodeManager for execution, until the maximum number of retries is exceeded.

For maple task scheduling, we prioritize scheduling to the same node where the input file is located (SDFS) to reduce network overhead. If the conditions are not met, we will schedule to the same node where the execution file is located (SDFS). If not, we will try to schedule any surviving node manager. When scheduling the juice task, we use hash shuffle, which randomly breaks up the intermediate and assigns it to different node managers for execution.

We encapsulate specific Maple/Juice task execution logic into a worker and provide a template file; the user only needs to fill in a Maple/Juice execution function and task execution-related configuration (input format, partition type, shuffle type, etc.). Like the user using Hadoop, the user provides the template file; the system will compile and upload it to the SDFS. NodeManager will obtain the executable file from SDFS and then fork sub-processes for execution.

3. Executables

Here's the conceptual design of the filter/join MapleJuice executables.

a) Filter: Maple reads input data as key-value pairs (kv), The provided logic filters and transforms the data based on a regular expression (regexCondition). If the data matches this condition, it's assigned a new key (generated randomly) and passed forward. The output is a new key-value pair where the key is determined by a random integer and the value is the original input data (if it matches the regex condition). The Juice simply combines all the kv pairs from the maple.

b) Join: Maple Function transforms each line of a dataset into a key-value pair. It reads a line, splits it into columns. And then uses a specified column as the key. It combines the dataset identifier with the entire line as the value. Juice Function Aggregates and combines data from two datasets. It segregates values based on their dataset identifiers ("D1" or "D2"). It then concatenates each pair of values from "D1" and "D2".

Experiments & Result Evaluation

We evaluated the performance of our MapleJuice along with Hadoop through a series of experiments designed to measure operational overheads and efficiency under various scenarios:

(i) **Filter1**: We measured the execution time of filter when processing a dataset with 100+ records (100MB in total) with regex condition "None". The following plots show that our system execution time is not linearly decreasing with the number of cluster's, a possible explanation is that we did not vary the numMaples=4 with respect to the cluster size to ensure stable running on both MapleJuice and Hadoop. As for the high stand deviation and execution time rise when dealing with 10 VMs, we suspect that it is due to high miss rate when scheduling tasks on 10 VMs regarding replication number is set to be 4 in both HDFS and

SDFS in our design.

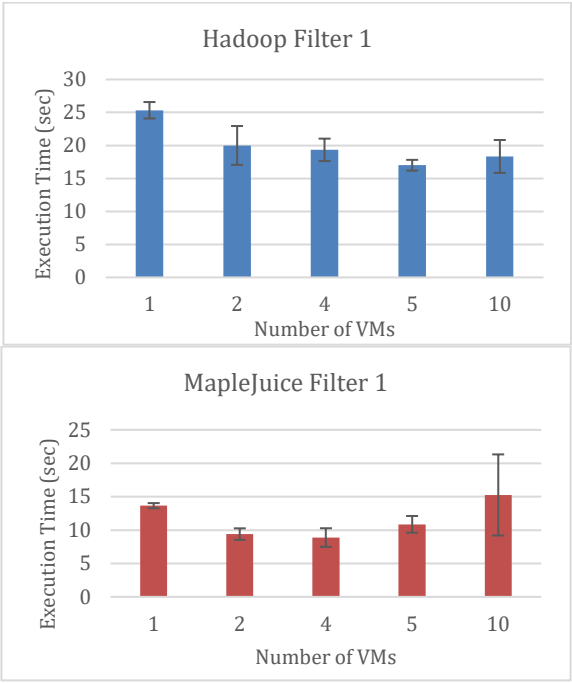

Figure 1. Filter "None" Execution Time

(ii) **Filter2**: We also conducted another experiment of filter on another regex pattern "N*", this time we witnessed a generally slower execution time compared with Filter1, it should be due to the increased number of matched lines.

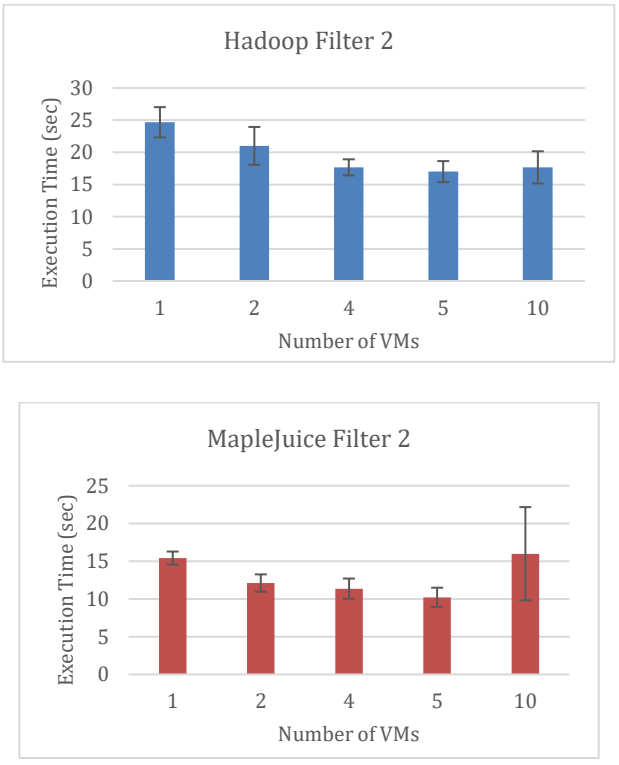

Figure 2. Filter "N*" Execution Time

(iii) **Join1**: For the join function, we use all the VMs to join the cluster, but we keep the numMaples=4 for both Hadoop and MapleJuice to ensure fair comparison. We varied the input dataset sizes by expanding one primary dataset with different length random strings as an additional column. We first select 2 datasets that have no overlap in the column values. The join functions relatively take longer time then filter, and for Hadoop, the amount of time processing each size of files grows slowly as a linear trend. We suspect that it is caused by relatively small row numbers in our testing dataset. Thus, the increase in unchecked random column length will not lead to very large time overhead when the file is locally available. However, it would still increase the time to transfer files between the peers when the files are not available locally.

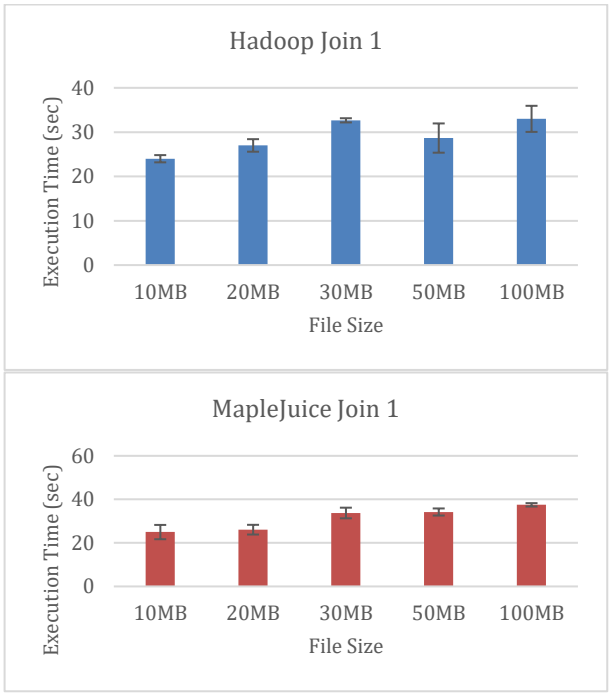

Figure 5. Join1 TimeElapse

(iii) **Join2**: Compared with Join 1, the Join2 generally takes slightly longer time to finish for most cases, which aligns with the intuition that it grows a little bit with increased number of operations.

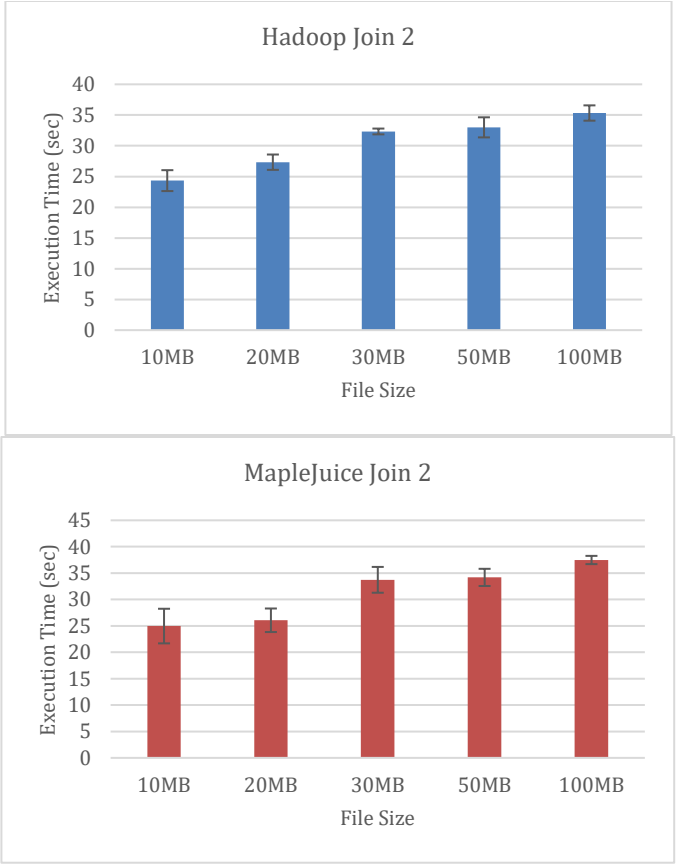

Figure 6. Join2 TimeElapse

Each experiment was repeated three times to ensure reliability, with plots showing average times and standard deviation. The discussion accompanying each plot will interpret the significance of these findings, examining trends and their conformance or deviation from expected outcomes. In general, our MapleJuice is not as stable as Hadoop, but it does work slightly faster when the number of clusters is close to the number of replicas in the SDFS.