CS425, Distributed Systems: Fall 2023
Machine Programming 4: MapleJuice+SQL

Released Date: Nov 8, 2023
Due Date (Hard Deadline): Sunday, Dec 3, 2023 (Code+Report due at 11.59 PM)
Demos on Monday Dec 4, 2023

The is a very intense and time-consuming MP! So please start early! Start now!
(Disclaimer: Like all MPs and HWs, all references to all companies and people in this spec are purely fictitious and are intended to bear no resemblance to any persons or companies, living or dead.)

ExSpace (MP3) miraculously merged with two social media companies Qwitter and KitKat to form a new conglomerate called…MartWall Inc. (Sidebar: if you have not seen the movie "Idiocracy", Indy highly recommends it. Relevant to this MP's joke is the "Costco" scene here: https://www.youtube.com/watch?v=sdNmOOq6T8Y . Anyway,…).

MartWall loved your previous work at ExSpace (and they're also aware of your great work on the mission to Saturn in HW3), so they've hired you as a "MartWall Fellow". That's quite prestigious! Congratulations!

You must work in groups of two for this MP. Please stick with the groups you formed for MP1. (see end of document for expectations from group members.)

MartWall needs to fight off competition from their three biggest competitors: Pied Piper Inc., Hooli Inc., and Amazing.com. They also have to fight the twin scourges of fake news and online shopping! So they've decided to build a batch processing system that is faster than Mapreduce/Hadoop.

You have three tasks in this MP:

1. Your first task at your job is to use the SDFS (from MP3) and the failure detector (MP2) to build a new parallel cloud computing framework called **MapleJuice,** which bears similarities to MapReduce.
2. Your second task is to implement two simple SQL-style queries **over** MapleJuice.
3. Your third task is to **compare** the performance of MapleJuice against the latest version of Apache Hadoop (this involves deploying and running Apache Hadoop on the VMs).

These tasks are sequential, so please start early, and plan your progress with the deadline in mind. We recommend budgeting 1 week for each of the 3 tasks above, and the fourth week for experiments and report writing. Please DO NOT start a week (or even two weeks) before the deadline – at that point you're already too late and will likely not be able to finish in time.

Below, MartWall has been very detailed about the design of MapleJuice. However, they want you to fill in some gaps in the design, and of course to implement the system. Be prepared to improvise, be prepared to deploy new systems, and be prepared for the unknown. Remember – Move Fast and Break Things! (i.e., build some pieces of code that run, and incrementally grow them. Don't write big pieces of code and then compile them all and run them all!)

MapleJuice shares similarities with MapReduce/Hadoop, except that it is simpler. You can look at Hadoop docs and code, and use a similar design as Hadoop/YARN, but you cannot reuse any code from there (we will check using Moss. Also it's faster to write your own MapleJuice than borrow and throw out code.).

In this MP you should use code from MP1, MP2, and MP3.

Interface: MapleJuice consists of two phases of computation – Maple (brother of Map) and Juice (sister of Reduce). Each of these phases is parallel, and they are separated by a barrier. MapleJuice is intended to run on an arbitrary number of machines, but for most of your experiments you will be using up to the max number of VMs you have. Any of the commands below must be invokable from any of the N machines.

MapleJuice is invoked via two command lines, but overall a MapleJuice job takes as input a corpus of SDFS files and outputs a single SDFS file.

As you'll see below, the Maple function processes *a fixed number of input lines* (from a file) simultaneously at a time (choose a number between 20 and 100), while the traditional Map function processed only *one* input line at a time. (These files must be stored in SDFS.) Beyond this distinction, the MapleJuice paradigm is very similar to the MapReduce paradigm.

Maple Phase: The first phase, Maple, is invokable from the command line as:

```
maple <maple_exe> <num_maples>
<sdfs_intermediate_filename_prefix> <sdfs_src_directory>
```

The first parameter `maple_exe` is a user-specified executable that takes as input one file and outputs a series of (key, value) pairs. `maple_exe` is the file name local to the file system of wherever the command is executed (alternately, store the executable in SDFS). The last series of parameters (`sdfs_src_directory`) specifies location of input files.

You must *partition* this entire input data so that each of the `num_maples` Maple tasks receives about the same amount of input. Hash partitioning is ok.

The output of the Maple phase (not task) is a series of SDFS files, one per key. That is, for a key K, all (K, any_value) pairs output by *any* Maple task must be appended to the file `sdfs_intermediate_filename_prefix_K` (with K appropriately modified to remove any special characters). This is another difference from MapReduce. After the Juice phase is done, you will have the option to delete these intermediate files.

Juice Phase: The second phase, Juice, is invokable from the command line as:

```
juice <juice_exe> <num_juices>
<sdfs_intermediate_filename_prefix> <sdfs_dest_filename>
delete_input={0,1}
```

The first parameter `juice_exe` is a user-specified executable that takes as input multiple (key, value) input lines, processes groups of (key, any_values) input lines together (sharing the same key, just like Reduce), and outputs (key, value) pairs. `juice_exe` is the file name local to the file system of wherever the command is executed (cleaner still, store it in SDFS). The second parameter `num_juices` specifies the number of Juice tasks (typically the same as the number of machines, but you could experiment with more tasks than machines).

You will have to implement a shuffle grouping mechanism – support both hash and range partitions. If you wish, this can be parameterized from your command line.

Each juice task is responsible for a portion of the keys – each key is allotted to exactly one Juice task (this is done by the Leader server). The juice task fetches the relevant SDFS files `sdfs_intermediate_filename_prefix_K's`, processes the input lines in them, and appends all its output to `sdfs_dest_filename.` You could also make your system keep the contents of `sdfs_dest_filename` always sorted by key (similar to SSTables we discussed in class).

When the last parameter above delete_input is set to 1, your MapleJuice engine must delete the input files automatically after the Juice phase is done. If delete_input is set to 0, the Juice input files must be left untouched.

Finally, implement both hash and range partitioning (shuffling). If you wish, you can give these as command line options.

Design: The MapleJuice cluster has N (up to max number of VMs) server machines. One of them is the leader server, and the remaining N-1 are worker servers. The leader is responsible for all critical functionalities: receiving maple and juice commands, scheduling appropriate Maple/Juice tasks, allocating keys to Reduce tasks, tracking progress/completion of tasks, and dealing with failures of the other (worker) servers. In short, any coordination activity, other than failure detection/membership, can be done by the leader.

To simplify your work, the MapleJuice cluster only accepts one command (maple or juice) at a time, i.e., while the cluster is processing maple (or juice) tasks, no other juice (or maple) tasks can be submitted. However, a sequence of jobs can be submitted (they will be queued to be executed one at a time; you can use FIFO across jobs). Additionally, you may assume that the leader server/RM is fault-free (though in reality, it's always wise to have a backup leader, for this MP we won't fail the leader).

Worker failures must be tolerated. When a worker fails, the leader must reschedule the task quickly so that the job can still complete. When a worker rejoins the system, the leader must consider it for new tasks. Worker failures must not result in incorrect output.

You must use the code for MPs1-3 in building the MapleJuice system. Use MP1 to log, MP2 to detect failures, and MP3 to store the input to and results from the MapleJuice topology.

Create logs at each machine (queriable via MP1 or via grep). You can make your logs as verbose as you want them (for debugging purposes), but at the least a worker must log each time a Maple/Juice task is started locally, and the leader must log whenever a job is received, each time a Maple/Juice task is scheduled or completed, and when the job is completed. We will request to see the log entries at demo time, either via local grep or the MP1's querier.

Other parts of the design are open, and you are free to choose. Design first, then implement. Keep your design (and implementation) as simple as possible. Use the adage "KISS: Keep It Simple Si…". Otherwise, MartWall Inc. may, in their anger at your complex design, fire you into space (or worse, into deep see near the Titanic), with only a book to read. That would be a very boring vacation, right?

We also recommend (but don't require) writing tests for basic scheduling operations. In any case, the next section tests some of the workings of your implementation.

SQL Layer: Implement the ability to support SQL-style operations, specified in the SQL format (this was discussed in class during Key-value store lectures, and you can look up online). Support (at least) the following two types of SQL queries (you can support more if needed):

1. Filter: *SELECT ALL FROM Dataset WHERE* <*regex condition*>. Condition should be able to specify any regular expression for that line, and the output should be all lines that match that condition.
2. Join: Given two Datasets D1 and D2: *SELECT ALL FROM D1, D2 WHERE* <*one specific field's value in a line of D1 = one specific field's value in a line of D2*>, e.g., *"WHERE D1.name = D2.ID" (you can program the Maple phases reading D1 and D2 respectively to know what D1.name and D2.ID are).*

3. Any of the VMs should be able to launch SQL queries (and MapleJuice jobs). The answers to an SQL query must be stored on SDFS, and the SDFS file name returned to the querying VM (at the end of the query).

In the MapleJuice design, think carefully about your design choices for the join. In particular, think about whether (a) sorting each data set would make the join faster, (b) replicating the smaller of the two datasets (you can assume you know the dataset sizes upfront) can make the join faster. Both Filter and Join above *must* be layered cleanly over MapleJuice, that is, you can only use MapleJuice stages, and not any other computations. (So you cannot use hacks like replicating the smaller dataset inside SDFS itself at every node in the cluster. Replicating via the normal Reduce key shuffling is acceptable.)

Dataset information appears later in this document (you may use others). Try to use datasets that are at least 100s of MBs large (if possible, even larger). Where datasets may not be available (e.g., applications 1 and 2 above), create (auto-generate) synthetic datasets that are non-trivial and use these in your experiments. Smaller is ok for tests, but ensure that the run time is at least a few tens of seconds, so that comparison makes sense.

Comparison against Hadoop: After you have your MapleJuice working, make it more efficient. Make it faster than Hadoop. Download Hadoop (latest version only) from http://hadoop.apache.org/ and run it on your VMs **(this step will take some effort, so give it enough time!).** Then layer your SQL queries (filter and Join) over Hadoop as well (for fair comparison, layer your own filter and Join implementation, but do NOT use Hive or Pig Latin).

Compare the performance of MapleJuice with Hadoop, and try to make MapleJuice faster. Most of the inefficiencies in MapleJuice will be in accessing storage, so think of how your files are written and read.

Make sure that you're comparing MapleJuice and Hadoop in the same cluster on the same dataset and for the same topology. To measure performance, you could use either completion time, or throughput (rate of jobs being completed), or both. Make the comparison *fair* -- Run the same job with the same settings for both systems. Are you able to beat Hadoop?

Datasets: Good places to look for datasets are the following (don't feel restricted by these):

- Champaign Map Databases are available at: https://gis-cityofchampaign.opendata.arcgis.com/search?collection=Dataset . Try queries like finding number of parking meters or number of apartment buildings with > 100 residents, etc. Look for other "GIS" databases.
- Stanford SNAP Repository: http://snap.stanford.edu/
- Amazon datasets: https://registry.opendata.aws/
- Wikipedia Dataset: http://www.cs.upc.edu/~nlp/wikicorpus/

Machines: We will be using the CS VM Cluster machines. You will be using all your VMs for the demo. The VMs do not have persistent storage, so you are required to use git to manage your code. To access git from the VMs, use the same instructions as MP1.

Demo: Demos are usually scheduled on the Monday right after the MP is due. The demos will be on the CS VM Cluster machines. You must use up to the max VMs for your demo (details will be posted on Piazza closer to the demo date). Please make sure your code runs on the CS VM Cluster machines, especially if you've used your own machines/laptops to do most of your coding. Please make sure that any third party code you use is installable on CS VM Cluster. Further demo details and a signup sheet will be made available closer to the date.

Language: Choose your favorite language! We recommend C/C++/Java/Go/Rust/Python.

Report: Write a report of less than 3 pages (12 pt font, typed only - no handwritten reports please!). Briefly describe your design (including architecture and programming framework) for MapleJuice, in less than 0.75 pages. Be concise and clear.

Show plots comparing MapleJuice's performance to Hadoop, for at least 2 different scenarios (involving datasets – you can pick from the above list or your own). For each dataset scenario:

1. (Filter 1, Filter 2) Pick TWO examples of Filter queries (one simple, one complex regex), and
2. (Join 1, Join 2) TWO examples of Join queries (one simple, one complex).
3. For #1 above, vary the cluster size (number of VMs) up to 10. For #2 above, vary data size (use all 10 VMs). Ensure you have at least 5 data points on the x axis.

(Please use the above names and numbers in your report) Since you are running both MapleJuice and Hadoop, there should be a total of EIGHT scenarios. Place the MapleJuice and Hadoop lines (for the same scenario and query) on the same plot, and write what you observe and why you observed it. Make sure you are comparing MapleJuice+SQL vs. Hadoop+SQL for the same setting (queries, cluster size, etc.) – compare apples to apples, not apples to oranges! Can you beat Hadoop on a majority of these cases? **Try to run jobs that take at least a few tens of seconds** (so that you can measure the differences in completion time). It is not required you beat Hadoop in all cases – in either case, discuss intuitively why you are seeing the performance differences you are seeing.

For each data point on the plots, take at least 3 measurements, plot the average (or median) and standard deviation. Run each experiment (for each system) on all the VMs in your allocation. **Devote sufficient time for doing experiments** (this means finishing your system early!).

Discuss your plots, don't just put them on paper, i.e., discuss trends, and whether they are what you expect or not (why or why not). (Measurement numbers don't lie, but we need to make sense of them!) Stay within page limit. Make sure to plot average, standard deviations, etc.

Submission: Submission instructions are similar to previous MPs (submit the Google form with your git link, submit the report on Gradescope, and signup for demo, and attend demo).

When should I start? Start now on this MP. Each MP involves a significant amount of planning, design, and implementation/debugging/experimentation work. Learning how to run any open-source system is also a time-consuming task, so please devote enough time to do this. Do not leave all the work for the days before the deadline – there will be no extensions.
Please note that the submission deadline is right at the end of Fall/Thanksgiving break – please plan appropriately, and **start early**.

Evaluation Break-up: Demo [60%], Report (including design and plots) [30%], Code readability and comments [10%].

Academic Integrity: You cannot look at others' solutions, whether from this year or past years. We will run Moss to check for copying within and outside this class – first offense results in a zero grade on the MP, and second offense results in an F in the course. There are past examples of students penalized in both those ways, so just don't cheat. You can only discuss the MP spec and lecture concepts with the class students and forum, but not solutions, ideas, or code (if we see you posting code on the forum, that's a zero on the MP).

We recommend you stick with the same group from one MP to the next (this helps keep the VM mapping sane on IT's end), except for exceptional circumstances. We expect all group members to contribute about equivalently to the overall effort. If you believe your group members are not, please have "the talk" with them first, give them a second chance. If that doesn't work either, please approach Indy.

Happy MapReduce (from us and the fictitious MartWall Inc.)!