CS425, Distributed Systems: Fall 2019
Machine Programming 4 – MapleJuice

Released Date: Nov 7, 2019
Due Date (Hard Deadline): Sunday, Dec 8, 2019 (Code due at 11.59 PM)
Demos on Monday December 9, 2018

YoureFired Inc. (MP3) just got acquired by the fictitious social media company SpaceBook Inc. SpaceBook Inc. loved your previous work at YoureFired (and they're also aware of your great work on the mission to Mars in HW3), so they've hired you as a "Spacebook Fellow". That's quite prestigious, congratulations!

You must work in groups of two for this MP (yes, Fellows also collaborate).

SpaceBook needs to fight off competition from their two biggest competitors: Pied Piper Inc., and Hooli Inc. They also have to fight the scourge of fake news. 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 write (at least) two **applications** using MapleJuice.
3. Your third task is to **compare** the performance of MapleJuice against 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. DO NOT start a week before the deadline – at that point you're already too late!

Below, SpaceBook 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! Or – Fail Fast+Often and Succeed Soon!

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.).

This MP requires you to 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 *10 input lines* (from a file) simultaneously at a time, 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 it in SDFS). The last series of parameters (`sdfs_src_directory`) specifies the location of the 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 Master 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 the them is the master server, and the remaining N-1 are worker servers. The master 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 master.

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). Additionally, you may assume that the master server/RM is fault-free (see extra credit below).

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

Use the code for MP1-3 in building the MapleJuice system. Use MP1 for debugging and querying logs, MP2 to detect failures, and MP3 to store the results from the MapleJuice topology (or input data if that's what you want to do).

Create logs at each machine (so that they are queriable via MP1). 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 master 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, via the MP1's querier.

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.

Applications: Write at least two applications using MapleJuice (variants of these have been discussed in class). Pick from among:

1. Wordcount (e.g., on the Gutenberg books),
2. For a directed input graph, output the Reverse Web Link graph (see SNAP repository),
3. For a web proxy log, output for each URL what percentage of access go to that URL (see Web Caching Datasets).
4. Map databases, e.g., 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.

Dataset information appears later in this document. Try to use datasets that are at least 100s of MBs large (if possible, even larger). Smaller is ok for tests, but ensure that the run time is at least a few tens of seconds, so that comparison makes sense.

Run your program using different values of num_maples (number of maple tasks) from the min to the max, and plot the resulting speedup on a graph. Each plotted data point must be the average of at least 3 experiment runs. Plot both the average (or median) and the standard deviation. Do the same for different numbers of juice tasks.

Comparison against Hadoop: After you have your MapleJuice working, make it more efficient. Make it faster than Hadoop. Download Hadoop from http://hadoop.apache.org/ and run it on your VMs (this step will take some effort, so give it enough time!). **Compare the performance of MapleJuice with Hadoop.** 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. Are you able to beat Hadoop in at least 1 out of the above 2 applications?

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.

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

- Stanford SNAP Repository: http://snap.stanford.edu/
- Web caching datasets: http://www.web-caching.com/traces-logs.html
- Amazon datasets: https://aws.amazon.com/datasets/
- 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.

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. Be concise and clear.

Show plots comparing MapleJuice's performance to Hadoop, for each of the 2 applications. Can you beat Hadoop at least 1 out of the 2 applications? 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!)

Extra Credit: Make MapleJuice tolerant to master failure, as much as possible. Specifically, when a master fails, the submitted job must not crash and executing Maple/Juice tasks must continue to completion (recall that SDFS writes and reads should work in spite of a failure). However, while there is no master (i.e., while re-election is still proceeding), job status may not be available and new jobs may not be schedulable on the MapleJuice cluster. The extra credit is not worth any points, but may be used at the end of the semester – if you're on a grade boundary, and did the extra credit, we may bump you up to the next higher grade.

Submission: There will be a demo of each group’s project code. Submit your report (softcopy) as well as working code. Please include a README explaining how to compile and run your code. Submission instructions are similar to previous MPs (see Piazza).

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.

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). SpaceBook Inc. is watching!

Happy Stream Processing (from us and the fictitious SpaceBook Inc.)!