CS 425 Distributed System MP4 Report

Group 43: Tzu-Bin Yan (tbyan2), Yeongyoon Park (ypark66)

I. Design

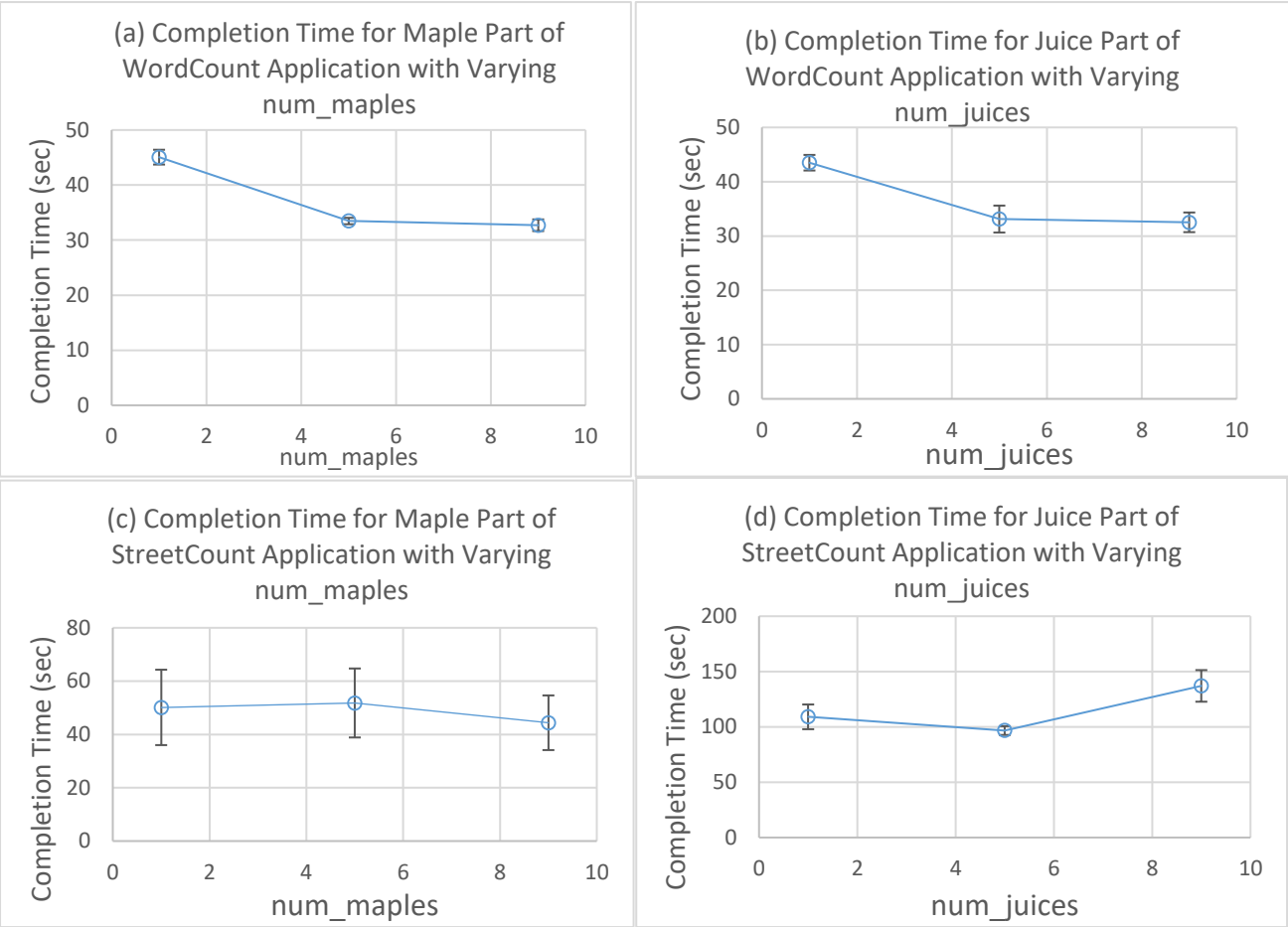
For this MP, the design of the MapleJuice System follows that specified in the spec and is composed mainly of three parts, i.e. the master, the workers and the clients. The master is the main component that receives requests (maple/juice jobs), queues them up (the master will perform the requests one at a time), retrieve data from SDFS and divide them if necessary (even division of work load among tasks based on number of lines of data in input files for maple tasks), assign tasks corresponding to the requests to workers (involving hash (done using MD5) or range partitioning of keys for juice tasks), combine the results and store them back to SDFS, and notifying the clients that the requests has been performed. The master also performs the reassigning of tasks if it detects failure of any of the workers. The master is designed to be started on a single VM (VM01) with the workers living on the other VMs. The workers receive assigned tasks and either directly process the input data that are contained in the assignment request for maple tasks (call maple exe on per ten lines of input), or retrieve the respective files for the specified prefix and key in the assigned task and process them for juice tasks. The processed results are sent back to the master for combining and storing into the SDFS. The clients are two small programs (maple.py and juice.py) that sends requests to the master containing request parameters as specified in spec. exe in maple and juice requests are in the form of Python scripts.

II. Performance Evaluation

We choose the word count application for counting words in a document for the first application and street count for counting number of address points per streets based on the Address_Points.csv file available at [1] for the second application.

For running program using different values of num_maples, we choose to run the maple part of the word count application with different number of num_maples on a generated 160MB file consisting of random words per line. For running program using different values of num_juices, we choose to run the juice part of the word count application with different number of num_juices on a generated 2.5 KB file with 370 different words spread out by line (we first call the maple part of word count on the 2.5KB input file and then run tests for num_juices based on the 370 intermediate files generated). The reason we used a different a file for testing different values of num_juices is due to juice on 160MB file's intermediate files getting finished too quickly (within 1 second) to perform any kind of analysis. Although the juice file may seem to be small, the run time still exceeded the required 10s of seconds, with possible explanation given below. The plotted results are provided in plot (a) and (b).

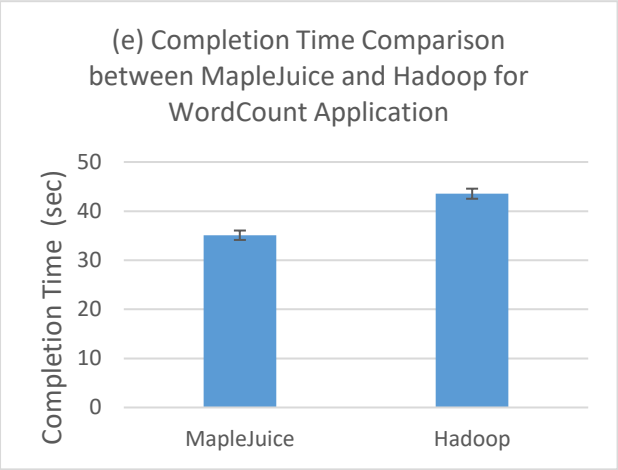
As we can see in (a), running with increasing number of maple tasks has a tendency of decreasing the time needed to perform a request. This is sensible since with increasing number of tasks, the level of parallelism is higher, which thus results in finishing processing the constant size input in less time. Nevertheless, observe that the decrease slows down when num_maples becomes larger. We believe the

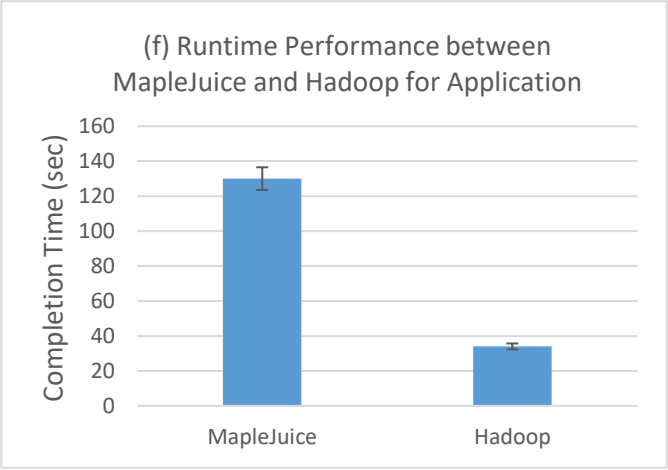


reason for this is due to the inefficiency in master design since the master will have to send the input to all workers running maple tasks respectively. This operation takes a long period of time when input is large and when there are enough maple tasks to divide the input, the bottleneck of the completion time becomes the time master needs to send all inputs, plus the time master needs to process all sent back results from workers, plus the time master needs to store all results in SDFS, therefore limiting the decrease in completion time that can be achieved.

As we can see in (b), running with increasing number of juice tasks also has a tendency of decreasing the time needed to perform a request. This is sensible with the same argument of increasing parallelism resulting in better performance. However, observe that there is also a slowdown in decrease of completion time when the num_juices increases. We believe the reason for this is due to an inefficiency in the SDFS design of MP3 since under our design the SDFS nodes serve any request one at a time rather than serve them simultaneously. This cast a limit on how many clients that can be served simultaneously by the SDFS at the same time to be roughly the number of SDFS nodes available in the SDFS. In this test, the SDFS system contains of 5 nodes, so it is expected that at most 5 MapleJuice workers are expected to be served by the SDFS at the same time when they try to retrieve intermediate files from the SDFS, which results in the parallelism that can be achieved to be limited and thus a slowdown in decrease of completion time. This inefficiency in SDFS also explains the fact that the completion time for the juice job took around 30~40 seconds even with such a small file, with most of the time wasted on workers waiting for the SDFS to serve their requests.

In addition to word count, we also did the same performance evaluation for the street count (shown in (c) and (d)), where for the maple it generally followed the same performance argument in (a) with small deviation possible due to network status. However, we observed a sharp increase in completion time for the juice phase for street count than that observed in (b). We believe that the reason for this is due to the inefficiency of SDFS (as explained for (b)) being amplified by the very high numbers of distinct street names in the Address_Point.csv file with there being a total of 825 streets in the file.





As for comparison with Hadoop, the performance is evaluated based on a cluster of 9 workers for both MapleJuice and Hadoop. For the two applications, the input is the generated 160MB text file and Address_Points.csv respectively. The Hadoop applications are based on the WordCount example file in MapReduce tutorial on [2] and their python counterpart is designed to do the exact same thing (the street count application modifies the WordCount code to parse out the StreetName field of the csv). The results are provided in (e) and (f) respectively.

As one can see in (e), for this task we have that MapleJuice in fact out-performs Hadoop slightly with there being an around 8 second runtime difference. We believe the reason for this is due to that in Hadoop, even if that the actually task running time is small, the time needed to initialize each worker to run tasks takes a very significant amount of time, therefore, resulting in Hadoop losing to MapleJuice.

For the second application (as shown in (f)), running on Hadoop was much faster than running on MapleJuice. One of the reason that Hadoop outperformed MapleJuice is due to the significant amount of time needed when running juice for MapleJuice. We believe that one of the main reason Juice was slow was due to the performance of our SDFS, where our workers were often left in a hold waiting for connection while trying to get the key file from the SDFS (the same inefficiency as in (b) and (d)). The overall performance of our MapleJuice did not reach our expectation. The design of the SDFS needs improvements to increase the parallelism for slaves when serving client requests.

III. Reference

[1] https://gis-cityofchampaign.opendata.arcgis.com/datasets/address-points

[2] https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html