CS425/ECE428 MP4 Group 41 Report

James Hsu chsu41, Zitong Chen zitongc2

Introduction:

In this machine problem, we implemented MapleJuice, a parallel cloud computing framework that is similar to Apache Hadoop's MapReduce.

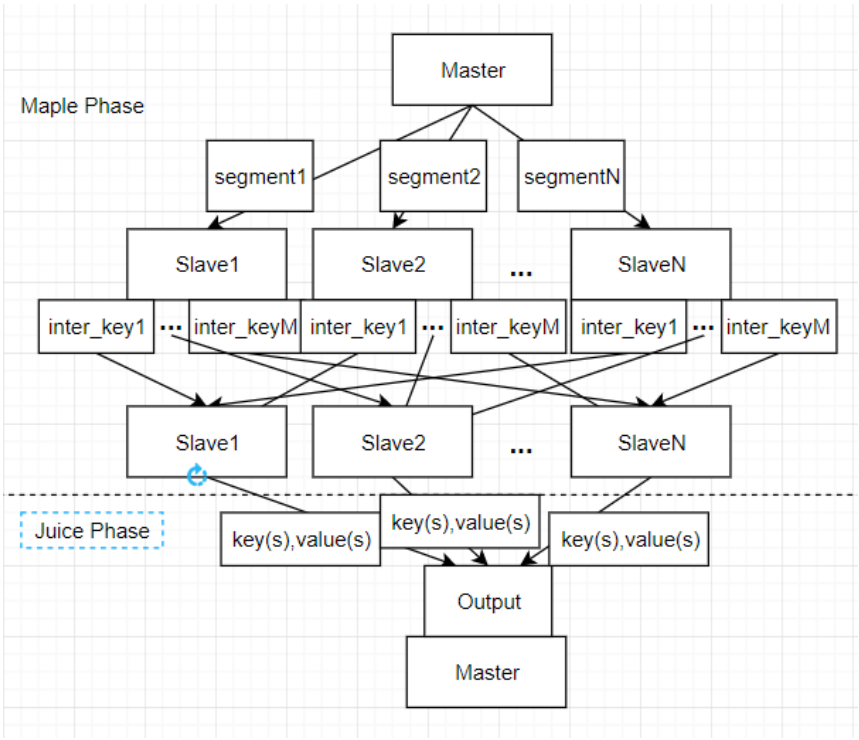
Framework:
Like MapReduce, MapleJuice consists of two phases of computation—Maple and Juice. The Maple phase of the application partitions the input file into *num_maple* pieces for each machine in the VM cluster that is participating in the computation. During the Maple phase, each VM outputs a (key,value) line for each key in its partition of the original file, and appends it into *sdfs_intermediate_filename_prefix_K*. The more keys there are, the more intermediate files there will be.

The Juice phase of the application, on the other hand, processes the *sdfs_intermediate_filename_prefix_K* files to combine all the values based on the keys to generate a *sdfs_dest_filename* file with all the final (key,value) pairs appended to it. The *num_juice* parameter specifies the number of juice tasks in that will be deployed in the VM cluster.

Design:
Our MapleJuice cluster, built on the SDFS that is implemented in our previous machine problem, works as follows:
Upon receiving a Maple request, the master node will distribute the job evenly across *num_maple* number of machines by splitting the input file into that many segments. Each node with a file segment will independently output (key,value) lines based on all the keys it reads in its segment, and append the (key,value) line into a shared file in the SDFS with filename *sdfs_intermediate_filename_prefix_K*, so that all the intermediate values will be aggregated.

Upon receiving a Juice request, the slave nodes, based on the hash value/range of the key of each file, will send the file to the correct node, where the final juice phase will be carried out. Each node in the VM cluster that is responsible for a juice task will process its portion of key files, sum up all the (key, value) lines in respective files, and output final (key,value) pairs into a single shared destination file.

The following diagram shows the dataflow structure of our MapleJuice framework:



Evaluation:

This section contains our test data. As we can see, the performance of our MapleJuice framework outperforms Apache Hadoop's MapReduce both for the word count and word frequency application. The speedup possibly comes from the fact that MapleJuice processes 10 lines at a time in the Maple phase, and file I/O doesn't become the bottleneck yet for the simple task of counting word occurrences. Also, both applications exhibit similar relationships between MapleJuice and MapReduce. This may be due to the similar nature between the application of word count and word frequency. The result is very much expected by us.

Environment: 10 machines

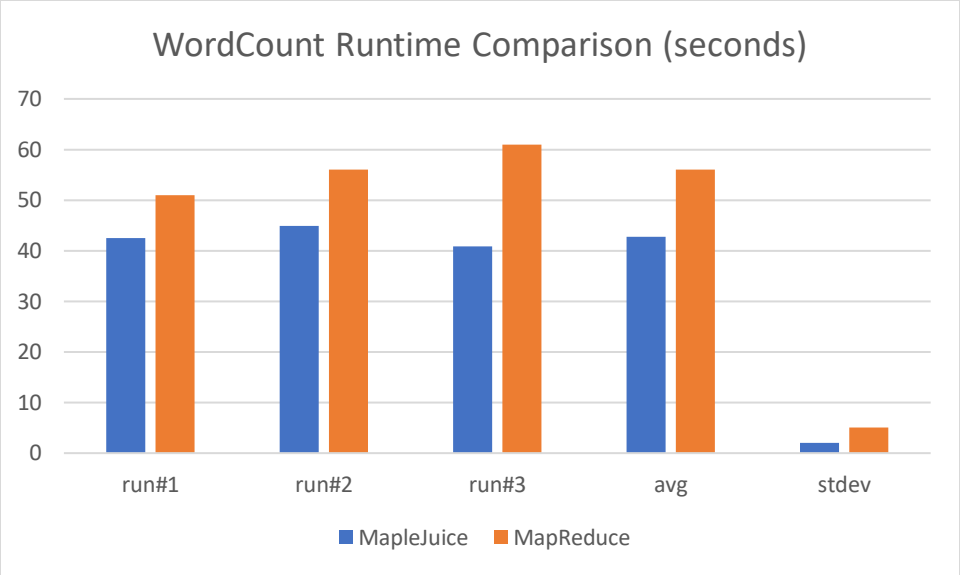
Word Count:

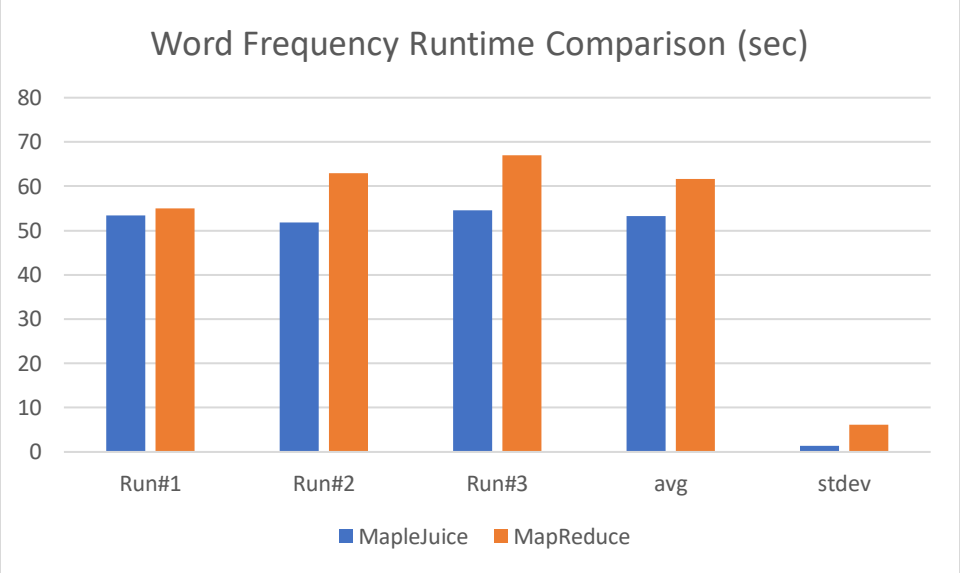
Input set : input.txt (Self-generated, 123M)

Word Frequency:

Input set: input.txt (Self-generated, 123M)

		MapleJuice	MapReduce
Word Count	Run#1	42.516s	51s
	Run#2	44.848s	56s
	Run#3	40.840s	61s
	avg	42.73467s	56s
	stdev	2.012928s	5s
Word Frequency	Run#1	53.4812s	55s
	Run#2	51.8801s	63s
	Run#3	54.547s	67s
	avg	53.303s	61.667s
	stdev	1.342s	6.110s


WordCount Runtime Comparison (seconds)
70
60
50
40
30
20
10
0
run#1
run#2
run#3
avg
stdev
MapleJuice
MapReduce


Word Frequency Runtime Comparison (sec)
80
70
60
50
40
30
20
10
0
Run#1
Run#2
Run#3
avg
stdev
MapleJuice
MapReduce