Distributed System MP4 Report

Group 52
Zhichang Chang(zchang8)
Lanxin Zhang(lanxinz2)

Program Design:

In this program, we add the logic for our MapleJuice based on the SDFS and failure detector from our MP2, and MP3. We use Golang built-in RPC library to build connections between each node, and we use channel to count the tasks from each node. We build two applications, WordCount and ReverseWebLink.

User can invoke Maple with maple_exe, num_workers, intermediate_file_prefix, and src_directory from any node, and this node would send the request to the master. The master would extract all the files from the src_directory and split them approximately evenly to the workers. Each worker starts its own Maple task with the specific function exported from Golang plugin. Instead of reading one line at a time, the Maple function would read 10 lines at a time. After its finished, the workers would send the (key, value) pairs result back to the master through the RPC between them. After all the workers finished Maple task, and master received all the results, it would put all values with the same key into the intermediate file with the intermediate_file_prefix entered by the user. However, there would be so many intermediate key files that if we put them all into the SDFS, it would cost too much time. Therefore, we split all the files equally into 10 chunks and put them in 10 folders, zip the 10 folders and put the result to the SDFS file system. In this way, it can largely save the time to put files in to SDFS, since there would be thousands of intermediate files.

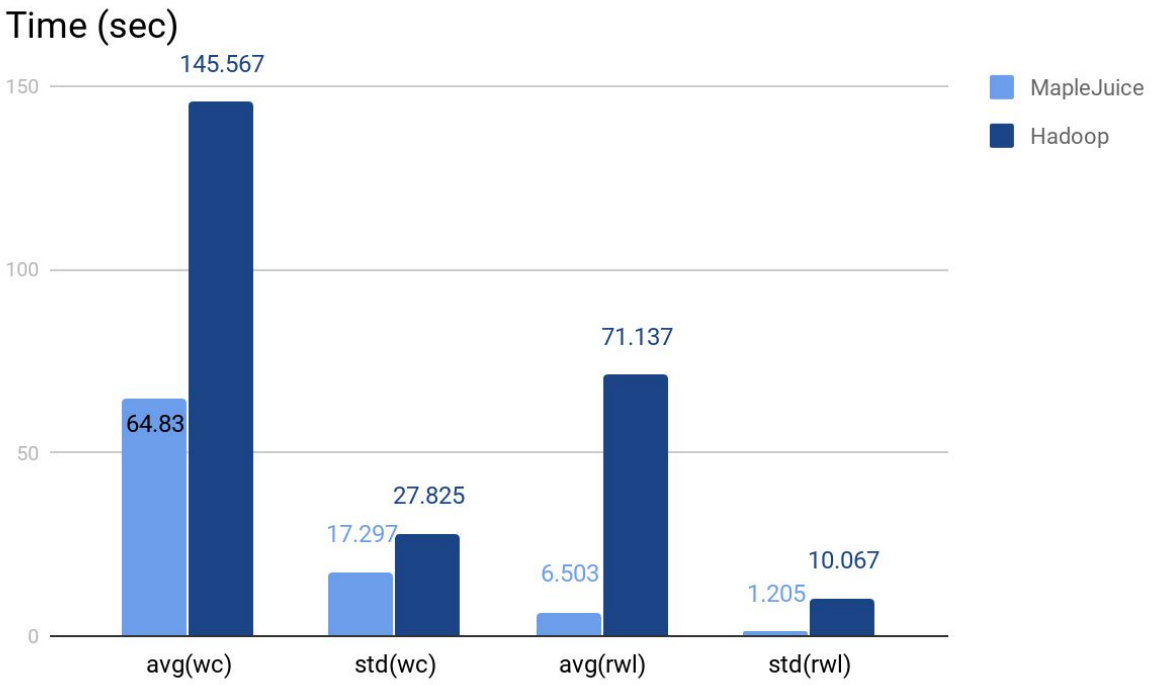
For the Juice phase, user can invoke it with juice_exe, num_workers, intermediate_file_prefix (must be the same as the one entered for Maple), dest_filename, and a delete_input to choose whether to delete the intermediate files produced during the Maple phase or not. After receiving the Juice request, master would equally distribute the 10 intermediate zip files to the workers. Then workers start their own Juice task by first extracting intermediate key files from the zipped folders, and then do the task with the specific Juice function exported from Golang plugin. After that, the workers write the (key, value) result with a map back to the master through RPC. When master received the result from workers, it write them to the dest_filename specified by the user.

MapleJuice vs. Hadoop

For each of the 2 applications, we take 3 measurements in seconds for both MapleJuice and Hadoop with all 10 vms. The input dataset for WordCount (wc) is 10 files each about 10MB size, and the input dataset for ReverseWebLink (rwl) is 10 files each about 100KB size.

	wc1	wc2	wc3	rwl1	rwl2	rwl3
MapleJuice	59.59	46.8	88.1	5.98	8.17	5.36
Hadoop	128.256	123.619	184.826	65.123	62.969	85.319

	avg(wc)	std(wc)	avg(rwl)	std(rwl)
MapleJuice	64.83	17.297	6.503	1.205
Hadoop	145.567	27.825	71.137	10.067



From the measurements, we can see that MapleJuice performs far better than Hadoop in both applications, which is what we expected. There are some reasons that helped MapleJuice to execute faster. We used RPC to build connection so that the workers can send the task results to master very quickly. We put all intermediate files into a few zipped folder and then put them in to SDFS file system, which saved us a lot of file transfer time. Also, in the Maple phase, we process 10 lines at a time while Hadoop processes one line at a time. Also, Hadoop takes some time to start its applications too, since even with small dataset, it still takes about 1 minute to execute.