CS 425 MP3: MapleJuice

Dayue Bai (dayueb2), Yitan Ze (yitanze2)

1. Design

In our implementation, one of the nodes in the MapleJuice cluster is the master node and the other nodes are worker nodes. Every node in the cluster is both a server and a client . The master node is responsible for processing maple and juice job requests sent from other worker nodes, scheduling tasks, shuffling keys for Juice tasks, and handling worker failures by rescheduling tasks. Our MP3 utilizes MP2 SDFS to store all files needed to perform Maple & Juice jobs. It uses MP1 Group Membership to provide membership service including worker failure detection. Our design assumes that the master node will never fail so that we can tolerate worker failures. We show how we design each functionality below.

Maple *<maple_exe><num_maples><sdfs_intermediate_filename_prefix> <sdfs_src_directory>*

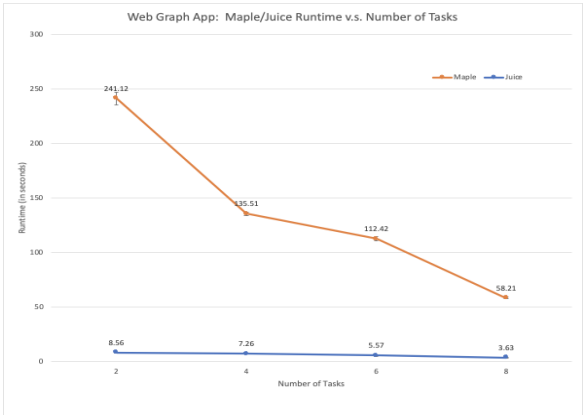
We upload all input files to SDFS and split input files by maple task number. For each maple task, we elect a worker node to handle the task based on **locality**. The more input files a node stores locally for a given task, the more likely it is the worker for the task. We schedule only one task for each worker at a time. Unscheduled tasks will be processed later when there is an available worker. When workers receive a task, it will download input files from SDFS and use <maple_exe> to process each input file (50 lines at a time) and send all the output to the master node. Then the worker becomes available. When the master receives a complete task from the worker, it will create a file per the key of task output and store all values associated with the key to the file. The master will also assign an unscheduled task to an available node upon the completion of a task. The master will upload all intermediate files to SDFS when all tasks are processed.

Juice *<juice_exe> <num_juices> <sdfs_intermediate_filename_prefix> <sdfs_dest_filename> delete_input={0,1} shuffle_option={1,2} // 1: HASH, 2: RANGE*

We first shuffle, partition and assign input files to tasks either using hash or range partitioning. Like Maple, the master will also elect a worker node based on **locality**. The master assigns Juice tasks to workers the same way as it does for Maple tasks. Upon receiving a task, the worker behaves the same way as it processes Maple tasks. Whenever a master node receives the output of a complete juice task from workers, it stores the output to the <sdfs_dest_filename> and schedules an unscheduled task to an available worker. The output file will be uploaded at the end.

Failure Handler: We design a dedicated thread used to reschedule incomplete tasks. When master's membership service detects a worker failure, this rescheduling thread will add tasks assigned to the failed workers to the **unscheduled task list**. Incomplete tasks will be scheduled when there is an available node.

2. Applications & Measurement



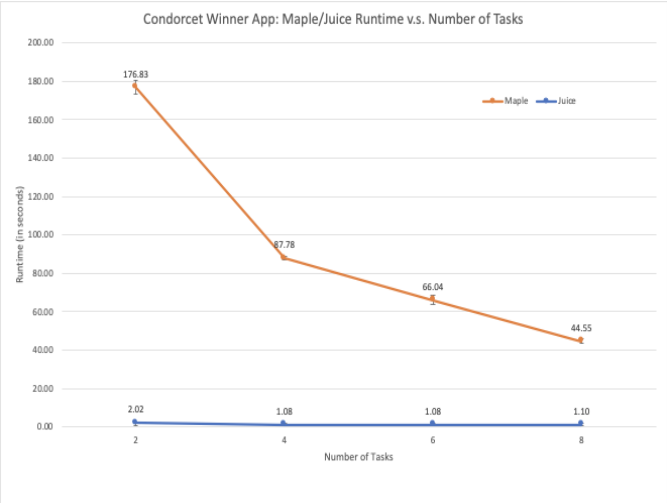
For both applications: Web Graph and Condorcet Winner, we used all 10 VMs to test the Maple and Juice performance given different numbers of tasks. We also compare the performance of MapleJuice against Hadoop using the same setting. The input file size for both applications are around 100MBs. We did 3 experiments for each test, and measured the mean and standard deviation for each point. *Note: we include stdev bar for each data point in all plots. Please zoom in to check*. It isn't obvious to see compared to data range.

(i) Web Graph Application

The Web Graph Application counts the number of links that points to the nodes with id ranges from 1 to 50. We use data from Stanford SNAP. For both Maple and Juice, we can

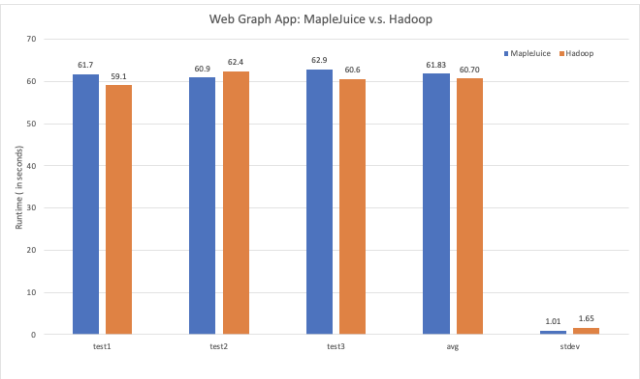
see that the run time decreases as we increase the number of tasks. Reason: we assign a task to only one node. When the task number increases, more nodes process tasks concurrently and each task contains less input files so that run time decreases. The Maple phase will generate at most 50 intermediate files with keys in the range: 1-50. Thus, Juice phase will process much less amount of data than the Maple phase. This fact explains why Juice runs much faster than Maple in this application.

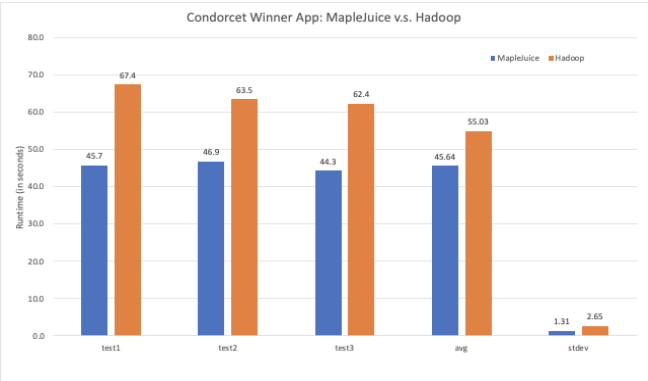
(ii) Condorcet Winner Application



The Condorcet Winner Application calculates the candidate that wins a majority of the vote against each of the other candidates given a large number of votes. The application used two chained MapleJuices. The first run of MapleJuice returns a single output file containing all non-repeating pairs of candidates where the first one dominates the second. The second run of MapleJuice takes the previous single output file as input, and calculates which candidate has the highest "dominate" count. The run time of the second MapleJuice is much faster than the first MapleJuice. Thus, the runtime of first MapleJuice determines the overall runtime efficiency. From the graph on the right, we can see that the runtime of Maple phase decreases significantly as the number of maple tasks increase for the same reason as we stated in (i)Web Graph. There is no obvious run time decrease in Juice phase when we increase the number of tasks from 4 to 8. Reason: the first Maple phase will only generate 3 intermediate files (given 3 candidates) and each file can only be assigned to one task (run on one worker node). Then, this application needs at most 3 Juice1 tasks. Therefore, increasing the number of tasks from 4 to 8 will not optimize the performance. Increasing the number of tasks from 2 to 3 will decrease the Juice run time a bit.

3. Comparing Against Hadoop





For each application, we also compare the performance of MapleJuice against Hadoop using the same setting and the same number (10) of VM. For Web Graph, the performance of Hadoop and MapleJuice are similar. However, for Condorcet Winner, our MapleJuice runs faster than Hadoop. It also runs faster than Web Graph. The reason is that the first Maple phase of Condorcet winner only generates 3 keys while the Maple phase of Web Graph can generate up to 50 keys. When the number of intermediate files increases, our MapleJuice takes much longer time to upload and download files. We searched on the web and found that Hadoop is designed for processing big data and its performance may be limited when processing small input files. For the Condorcet Winner application, the size of the first Map input is large but the size of output of Map (also the input & output of first reduce) is small.