MP5 – Maple Juice – CS425 T3 (3 hours)

Kevin Peters (klpeter2)
Jack Montag (montag1)

Design

For *Maple-Juice*, we decided to keep *SDFS* (Simple Distributed File System) in place and build *Maple-Juice* over it. *Maple-Juice* has *N > 0* processes where one is designated as the *master process* that's in charge of delegating *maple* and *juice* commands to other processes on the network. It is also responsible for introducing new processes to the network and ensuring the consistency of membership and file lists. The file replication protocol requires that all files on *SDFS* be replicated on *N / 2 + 1* processes at all times. This increases the chance that files will be available on *SDFS* and is efficient, because the files are replicated on just one more than half of the network's processes.

The implementation of *Maple-Juice* is straight forward. Any client on the network may issue a *maple* or *juice* command. When a *maple* command is issued, the issuing process will execute the command (on multiple threads if necessary) and notify the *master process* to *put* the resulting output file or files onto *SDFS*. Output files are *key-value* pairs in the form of *two-tuples*. The exact output depends on the executable specified in a *maple* or *juice* command. After a *maple* command has been performed and the output files are on *SDFS*, then a *juice* command can be performed. When a *juice* command is issued, the *master process* is first notified. The *master process* then delegates the work to a number (*M*) of processes on the network that a client has requested (*M <= N* and *M > 0*). The *juice* command is then executed at the delegated processes, and the *master process* is notified of each output file or files. The output file(s) are then *put* onto *SDFS*. If at any time one of the processes (except the *master process*) crashes during a *juice* execution, the *master process* will wait a certain amount of time and then reschedule the task to another process. For this particular implementation, we hard coded a timeout of 10 seconds. It is important to note that our implementation requires the *master process* to be running in order for *juice* commands to be handled correctly, but not necessary for *maple* commands.

The *maple* and *juice* executables we provided implement word counting (*wcmaple* and *wcjuice* respectively) for a file or files. The executables are completely independent of the distributed system; however, *Maple-Juice* may use them in a distributed manner. The executables are invoked at each process when a *maple* or *juice* command is received. The *wcmaple* executable takes a file of strings as input and outputs a file or files that contain *key-value* pairs; each *key* gets its own file. Anytime an occurrence of the *key* is found in the input file, the *two-tuple* (*key,* 1) is appended to its corresponding output file. After a *maple* command is issued and the file(s) are on *SDFS*, a *juice* command may be issued.

The *wcjuice* executable takes a file of *key-value* pairs as input and outputs a file that contains a *key-value* pair, where the *key* was the one found in the input file, and the *value* is all of the *values* of the input file summed together (word count). In the context of the distributed system, the *master process* is in charge of assigning *juice* tasks to processes. After each process has completed its *juice* task, the *master process* then takes the output file(s) of *key-value* pairs and merges them into a single output file.

Implementation and Usage

To compile the project, navigate to "mp5/Debug" and type "make all" in the console to build it. To begin running the distributed file system, log into a computer that you would like to be the *master process* (this is synonymous with the *introducer process* from MP3). In the main.cpp file in the "mp5/src" folder, there is a constant declared at the top of the file named "INTRODUCER_IP". Its value must be set to the machine that you'd like the *master process* to be. The *master process* must be running for any new process to join the network. So for example, if I logged into "linux6.ews.illinois.edu", I'd change the "INTRODUCER_IP" constant to be equal to "linux6.ews.illinois.edu" (default). Now any other process can join the network by connecting through the *master process*.

To begin running *Maple-Juice*, the executable "mp3" takes 1 argument: the ip-address of the current machine you're on. So for example, if I'm on machine "linux6.ews.illinois.edu", to start the exe I would type "./mp3 linux6.ews.illinois.edu". Each process will print out to the console and to a log generated in the local directory information whenever a process connects, times out/fails, a file is deleted, added, or command is performed. To kill a process for testing purposes you can press "Ctrl + c" to kill the process; it will be removed from the network after a short time.

Commands

1. get <filename> // Get a file from the distributed file system
2. put <filename> // Put a file on the distributed file system
3. delete <filename> // Delete a file from the distributed file system
4. generatefile <filename> // Generate a file with random strings
5. removefile <filename> // Remove a generated file
6. maple <maple_exe> <sdfs_inter_filename_prefix> <sdfs_src_filename_1> … <sdfs_src_filename_N>
7. juice <juice_exe> <num_juices> <sdfs_inter_filename_prefix> <sdfs_dest_filename>

An Example Run-through (*Maple-Juice*: Word Count)

Log into linux6.ews.illinois.edu. Navigate to the "mp5/Debug" folder and run the exe by typing in "./mp3 linux6.ews.illinois.edu" to start the *master process*. Then log onto another machine and do the same, making sure to type in its correct ip-address when starting the exe. We've supplied two sample input files named "mapleinput1" and "mapleinput2" for this example. We recommend following the naming conventions used below, as we've included a bash script named "cleanupfiles" that can be used to delete the files after a *Maple-Juice* run-through is completed.

1. *Put* the files on *SDFS*: "put mapleinput1 maplein1" and "put mapleinput2 maplein2"
2. Run *maple* on any machine: "maple wcmaple mapleout maplein1 maplein2"
3. Run *juice* on any machine: "juice wcjuice 3 mapleout juiceout"
4. *Get* the *juice* output: "get juiceout juiceoutlocal"
5. Close processes and run the cleanup script: "./cleanupfiles" to delete the output files from the /tmp/ directory.

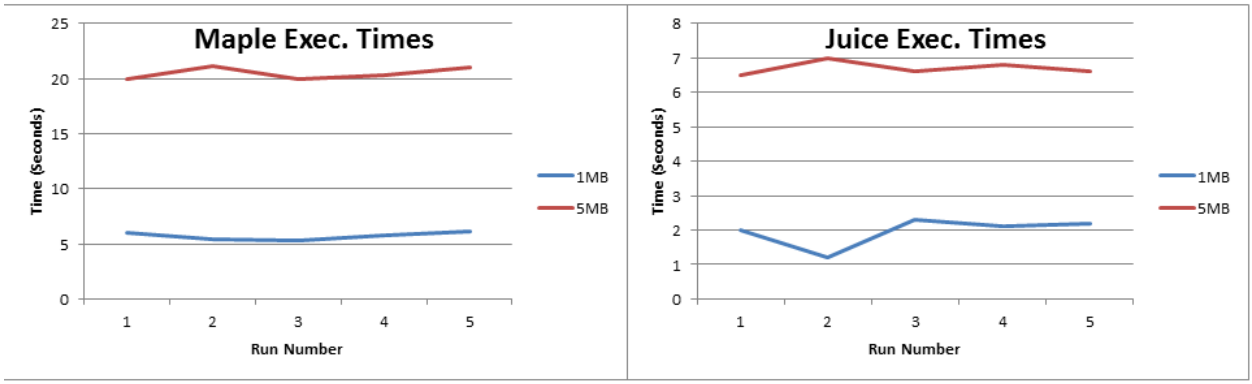
Note: ***The underscore character isn't allowed in file names!***
We also assume that each run of *maple* and *juice* will have unique output filenames associated with them.

Analysis

(i) It took less than a second to run *maple* and *juice* for 4 empty files on 3 machines.

(ii)

Maple and Juice Times to Complete Execution for Various File Sizes (N = 3, Files = 4)					
Maple	**1MB**	**5MB**	**Juice**	**1MB**	**5MB**
1	6	20	**1**	2	6.5
2	5.4	21.1	**2**	1.2	7
3	5.3	19.9	**3**	2.3	6.6
4	5.8	20.3	**4**	2.1	6.8
5	6.1	21	**5**	2.2	6.6
avg.	5.72	20.46	**avg.**	1.96	6.7
std. dev.	0.290975371	0.456800467	**std. dev.**	0.358701361	0.163299316



Extra Credit

In order to make our distributed file system fault tolerant of the master, we have designed a reliable master election system. Since the underlying distributed group membership system implements a total ordering protocol, all of the membership lists and file lists that exist on each of the processes on the network are synchronized. We decided to leverage this and also use this method to “elect” a master process when it fails. In other words, since the membership lists are guaranteed to be synchronized across all processes, we can simply take the next running process on the membership list after a master fails. A process that has dropped the network will be detected in 5 seconds (mp3 protocol), so it takes a little longer than 5 seconds for a new master to be “elected” and synchronized across all processes. The file list is also handled in the same way. It is synchronized across all processes and the new master process updates it periodically. When the new master process sees that a failure has occurred, it will replicate the necessary files to maintain our $N / 2 + 1$ protocol as necessary on any randomly selected processes that don’t already have the file.