CS425 MP3 report

Ruian Pan(ruianp2), Carl Zhang(hz5)

Design & algorithms:

For this distributed filesystem, SDFS, we use UDP protocol to send the membership list commands and file-system commands, and we use TCP protocol for sending files.
We used mp1 extensively for debugging, mainly the get-versions debugging and get debugging, where we compare the files easily. We based the mp on our mp2, which means the dissemination is based on gossip protocol, and the failure detection is based on SWIM style failure detector design.
We store 4 replicas of each SDFS file, and we use quorum for reading and writing. We use write quorum of 4 and read quorum of 1, so that we can accomplish: (1) write and write always intersect (2) read and write always intersect (3)Upon failure of three machines, there is always a newest version remaining.
We use a master for all the commands transfer. Filesystem commands, including put, get, delete and get-versions, are done by master. Master does the commands in the order it receives them, thus fulfilling total ordering. Master keeps track of the whole filesystem by a hashtable called file_info. Upon put command, master select four machines according to their current load, and sends their address back to client. Client then sends the files to them as replicas. Upon get command, master sends back a message consisting nodes where the requested files are stored to client, who then get the file from them.
We store the files in a SDFS folder for clarity. This filesystem is a versioned filesystem, and we store the previous versions inside the SDFS folder, namely, SDFS/filename. The versions are stored with the timestamps concatenated after the filename, and each machine keeps track of the versions of each file. Upon get-versions, master choose a random replica machine of the file and require a number of versions. The machine then get from its corresponding folders the number of files and send them back to the requesting machine.
For cases of failure, if a non-master machine fails, upon detection of master, the master would check which replicas are stored on the machine and redistribute them immediately. If master fails, the machines would automatically change a new master, based on a pre-numbered priority list. The membership list is very synchronized between the machines, so they would be able to check the membership list and independently select the highest one on our priority list as the master. The strategy for re-replication is as follows: after all failures converge, the master will reroute new node to store the new replicas based on load, and will send request to the non-failure replicas to send the files to these new nodes.

Measurements and plots:

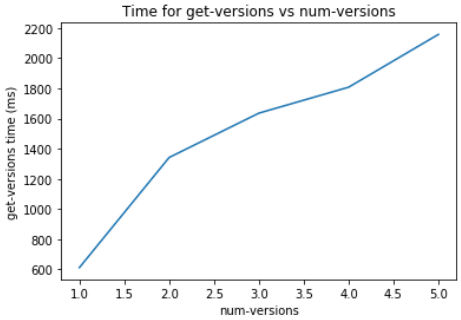
1. Re-replication time and bandwidth upon a failure (for a 40 MB file):

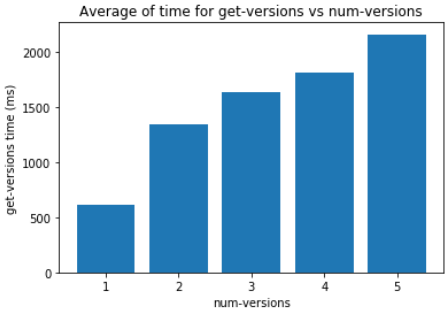
Re-rep time	354ms
bandwidth	105MB

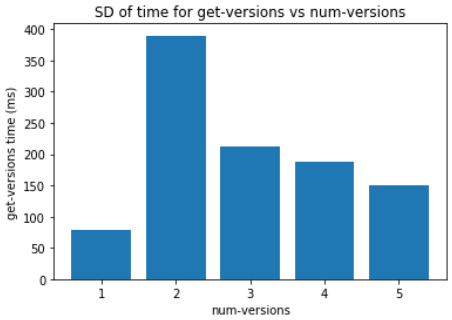
2. times to insert, read and update file of size 25 MB and 500 MB:

	insert	get	update
25mb	512ms	205ms	496ms
500mb	6532ms	1740ms	6177ms

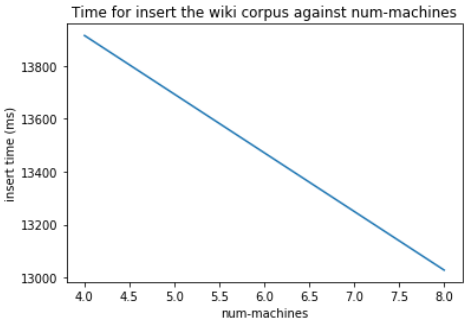
3. Plot of time of get-versions vs num-versions (for a 60 MB file):

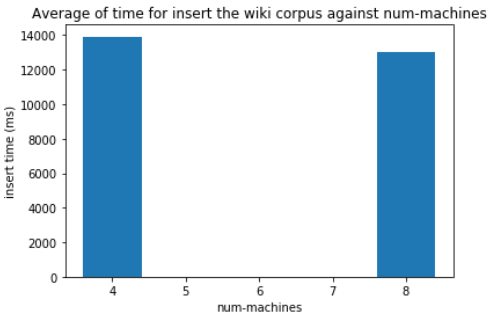


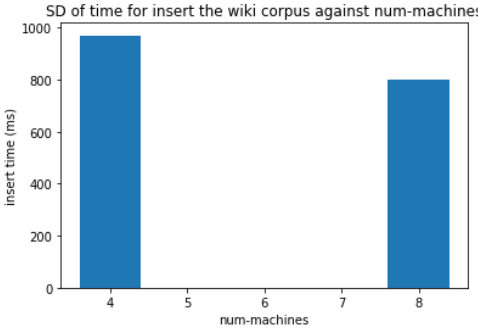




4. Plot of time of inserting Wiki corpus with 4 and 8 machines:







Analysis: There is an obvious linear relationship between the number of versions to get and the get-versions time, which makes sense since getting more versions means more files to transmit. The time of inserting wiki corpus is basically the same regardless of 4 or 8 machines, which makes sense since we are storing the same 4 replicas in either case.