CS 425 - MP3

Alex Huynh (ah10), Kaimeng Zhu (kaimeng2)

Design

The design of the system generally works as follows. A client will send a request to the leader, and the leader will coordinate what needs to be done. For example, if storing a file, the leader will choose a random set of four data nodes to store the file received from the client.

There is a group of four leaders, one active and three standby. Only the active leader does coordination. The network starts with one active leader. As more nodes join the system, the active leader will automatically promote nodes to be standby leaders, until four total leaders are reached. Four leaders are needed for the system to be fault-tolerant up to three failures. If the active leader fails, the next leader (sorted by ip address) will become the active leader and re-elect a new standby at random.

Files will be replicated four times to ensure fault-tolerance of three failures. If there are three failures, at least one replica will still be alive. In other words, $W = 4$. For fast reads, this algorithm has $R = 1$.

Past MP Use

MP2 was extremely useful, as it allowed the leader to figure out which machines were available to store the files. The extension of MP2 was doing arbitrary leader election, since this assignment does not allow for a hard-coded introducer.

MP1 can be used to see how the leader communicates with the data nodes. This is useful to see the general pipeline function, and give a blackbox view of who the active leader is. The difference would be to switch out grep functionality with data storage and retrieval.

Measurements

i) Re-replication time and Bandwidth 40MB file

	File size	Actual file size	Time (ms)	Time (s)
Trial 1	39845902	39845918	860.615204	0.860615204
Trial 2	39845902	39845918	728.430806	0.728430806
Trial 3	39845902	39845918	671.380225	0.671380225
Trial 4	39845902	39845918	691.289481	0.691289481
Trial 5	39845902	39845918	638.168837	0.638168837

Average	39845902	39845918	717.9769106	0.717976911
Std	0	0	86.17970366	0.08617970366

The ‘actual file size’ column accounts for the header included in GRPC messages. With these calculations, the re-replication time is about 0.718 seconds for four replicas, and the bandwidth is 277487461 bytes/second, or 277.487461 MB/sec.

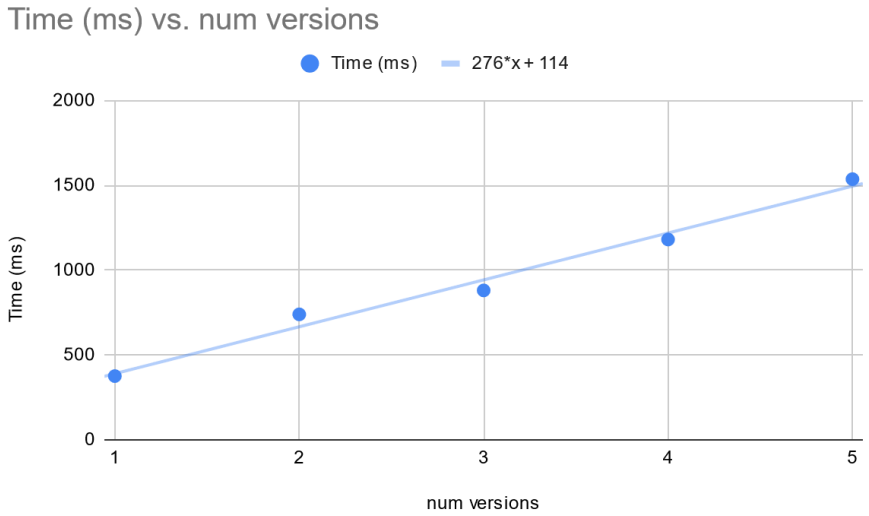
ii) Time to Insert, Read, and Update 25MB, 500MB file

	25 MB			500 MB		
	Insert	Read	Update	Insert	Read	Update
Average (s)	0.1104948916	0.4314570626	0.573161934	77.22943991	40.2304776	50.77111581
Std	0.01174767536	0.08520190903	0.04014539061	2.703336946	9.366349465	7.813735594

These results were quite interesting. Because this system uses R = 1, only one replica had to be sent when reading. However, it looks like the read operation took almost as much time as the Update operation (which sends the file to four machines sequentially). The expected result would be ¼ of the update time - about the expected result for the insertion time as well, but that is not what the results show.

It’s also interesting to see the 500 MB file took longer to insert, but was quite fast to update and read relative to its insertion speed. This could be due to network latency at the time the tests were conducted, but the exact cause is unknown.

iii) get-versions vs num-versions plot



The data for this plot was taken as follows: `get-versions [num-versions]` was called 5 times per `num-versions`. The average was then graphed.

The trend is about as expected - more files/versions being fetched lead to a linear increase in time to fetch said files.

iv) Time to store entire Wikipedia corpus

	Machines	
	4	8
	Time (s)	Time (s)
Trial 1	510.1493512	535.9893512
Trial 2	478.7837119	601.5237119
Trial 3	507.7734823	481.9334823
Trial 4	544.1221818	511.8221818
Trial 5	487.9708561	423.3708561
Average	505.7599167	510.9279167
Std	25.19499587	65.84715141

There is little to negligible difference between using 4 machines or 8 machines. This is highly likely due to the fact that this algorithm does not use any form of sharding. Therefore, it is almost as if the extra 4 machines are not even present. The results here are in line with expectation.