MP3: Distributed File System

Group 38: Rahul (rahulsk2) & Venkat (vn7)

1) Design: Algorithm Used & replication strategy

From the problem statement, it is clear that we need to store a total of 4 instances of a file in order for the system to be fault tolerant. We do not shard our files. We want W + R = 5, and since W = 4, R can be 1. Since we assume a reliable failure detector, we use a simplified version of the bully algorithm (incidentally preferring minimal IDs rather than maximal ones, though this does not impact the design in any way) where each node determines the next leader when the previous one fails deterministically. The leader will contact the other nodes to get the details of what files they have when it gets elected. We use passive replication for a more responsive system. Reads and writes occur between peer nodes directly, with the master node only performing the bookkeeping (we opted for this design choice because our master would be quite unresponsive during large file transfers otherwise). We do not perform any caching, because we do not broadcast a write log in order for a read cache to be kept up to date. We use MP2's membership list as a reliable list of nodes that are alive in order to pick suitable substitute replicas, and nodes to contact for reads. We do not use any explicit quorums, as we have W=4 and R=1.

2) How useful MP1 was for debugging MP3

We used our MP1 code regularly to identify errors while coding by checking for our debug output.

3) Measurements:

(i) re-replication time and bandwidth upon a failure

Bandwidth usage on failure: 40.01 MB
Re-replication time:
Average Time: 43.6 ms
Standard Deviation: 3.04 ms

Due to the large size of the file, our TCP packets stay close to maximal capacity. Consequently, we only see a bandwidth overhead of about 1% due to TCP.

(ii) times to insert, read, and update, file of size 25 MB, 500 MB (6 total data points), under no failure

25 MB:

25 MB (ms)	Insert	Updates	Delete
standard dev	38.3755426020271	18.8367213176816	0.146526448124562
average	459.412	496.168	1.818

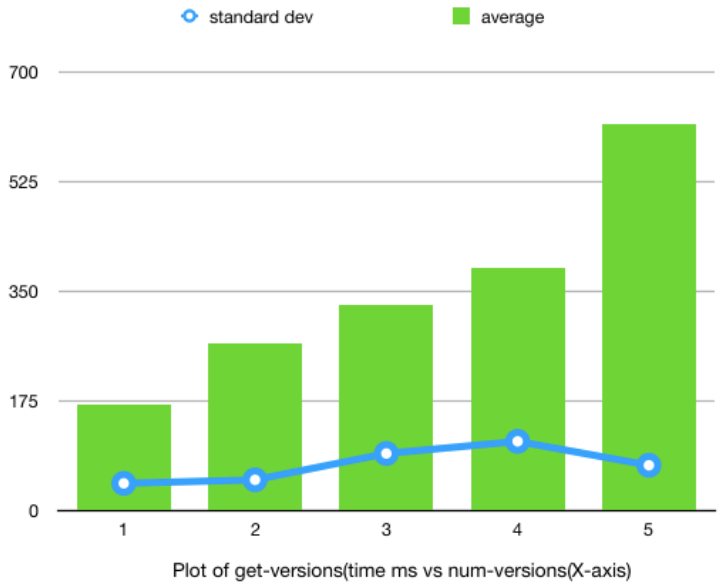
500 MB:

500 MB	Insert (s)	Updates (s)	Delete (ms)
standard dev	3.39337147981178	5.86225042112669	0.659295078094778
average	21.248	16.646	2.232

We can see that the statistics for inserts and updates are quite similar, since they perform effectively the same operations. The only difference between them occurs at the master, who must pick 4 replicas for an insert instead of looking them up for an update.

Because the file operations required to delete a file are independent of the size of a file, the time for deleting a file does not change much with an increase in file size.

(iii) Plot the time to perform get-versions as a function of num- versions



Plot of get-versions(time ms vs num-versions(X-axis)

This was done on a 25MB file

We can see that the time taken is roughly linearly related to the number of versions requested. However, the standard deviation does not change too much, as we usually query the same node for successive versions, and as such have a fairly predictable network speed.

(iv) time to store the entire English Wikipedia corpus into SDFS with 4 machines and 8 machines

8 machines:
Average time : 104.8 seconds
Std-Dev: 6.14 seconds

4 machines:
Average time : 98.8 seconds
Std-Dev: 2.59 seconds

We can see that the average does not change much, because the number of replicas is constant. However, the standard deviation does increase with the number of machines, probably due to the increased network usage.