CS 425 – MP2 Report

Group 69 – mgoel7 and bachina3

DESIGN AND ALGORITHM

Our SDFS is built on top of an asynchronous design paradigm where each server handles every request in a single event loop. This significantly reduces performance overheads usually incurred with threads while ensuring high level of concurrency.

In our implementation, we have designated one node as the leader which coordinates all the actions of the other nodes. It also keeps track of all the files in the system and where they are stored. This leader is dynamically elected by using a multicast-based election protocol with the node having the highest ID being designated as the leader every time. Furthermore, this leader node also acts as an introducer node which helps other nodes join the network. To make sure that a new node knows who the current leader/introducer in the network is, we have implemented a separate process to be dedicated as a DNS server which lets the new node know of the introducer's host and port. This DNS server is not part of the network and we assume that this process is always running.

In order to handle 3 simultaneous failures, we have set the number of replicas to be 4. This ensures that at least one replica is alive before re-replication for that file takes place and the system converges to an ideal state. In addition, we have designed our system to be read heavy. Thus, we have set W = 4 and R = 1, i.e., for write operations the leader will wait for ACKs from all the replicas and for reads it will just wait for ACK from any of the replicas. The following depicts how we handle each operation:

1. **PUT** - To initiate a PUT command, the client sends the request to the leader node. The leader node hashes the filename and generates IDs of 4 random nodes from it. If the file already exists in the system, then the replicas are the ones that already have that file. It then sends a DOWNLOAD request to the replicas, instructing them to download the file from the client node. At the same time, the leader also sends a PUT_REQUEST_ACK to the client to let it know that its request is being processed. On the replicas, the file transfer begins when they receive the DOWNLOAD request. Once they have downloaded the file, they send a DOWNLOAD_SUCCESS message to the leader which then logs the response. As soon as the leader receives DOWNLOAD_SUCCESS from all the replicas, it replies back to the client with a SUCCESS message.
2. **GET/GET-VERSIONS** – During the GET command, the client node checks if it has the file in its SDFS store. If yes, it simply copies the file from SDFS directory to user specified location. Otherwise, the client node sends the GET request to the leader node. The leader node responds back with the IDs of the machines where the file is stored. The client then tries to iteratively download the file from the replicas. As soon as file is downloaded from any one of the replicas, the client stops and displays a SUCCESS message to the user.
3. **DELETE –** To initiate a DELETE command, the client sends the request to the leader. The leader then sends an ACK to the client and also sends a DELETE_REQUEST to the replicas who have that file. Once the leader receives an ACK from all the replicas, it sends SUCCESS message to the client.
4. **ELECTION –** Whenever any node detects that the leader node has failed, it initiates the election phase and starts sending the ELECTION message to its pinging nodes. If a node receives an ELECTION message from any other node and it has not started its own election phase, it also starts its election phase and ignores any subsequent ELECTION messages received. In this way, all the nodes receive and send the ELECTION message. Furthermore, since each node as a full membership list, on each

receipt of the ELECTION message each node checks if it's the highest ID node in the current membership list. If yes, that node multicasts a COORDINATE message to every other node and waits for N-1 COORDINATE_ACKS. On receipt of a COORDINATE message, each node stops their election phase, set the new leader and send the COORDINATE_ACK message. Once a leader is elected, its address is also updated on the introducer DNS process.

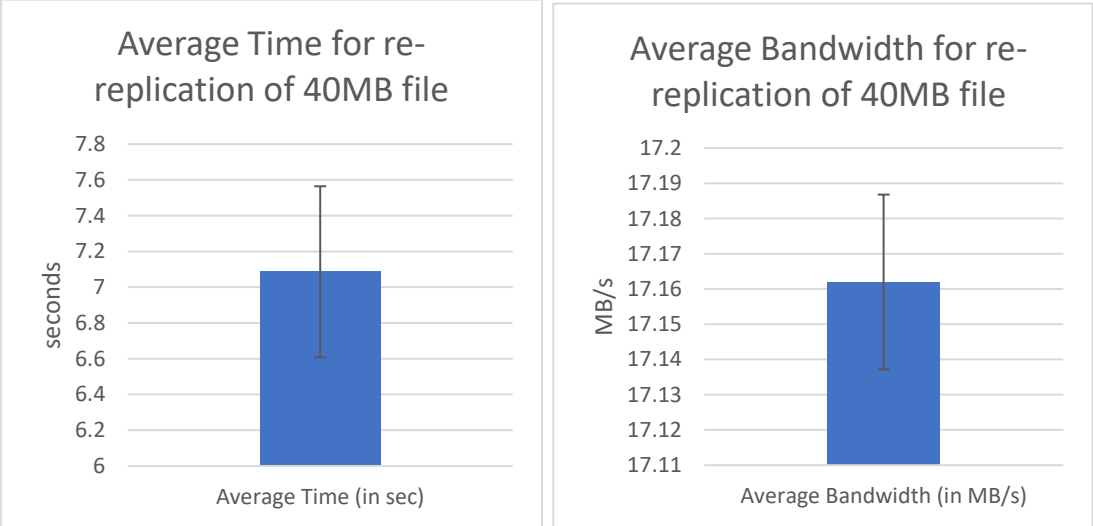
We have utilized asynchronous scp commands for secure and optimized file transfers between the nodes.

USE OF PREVIOUS MPs

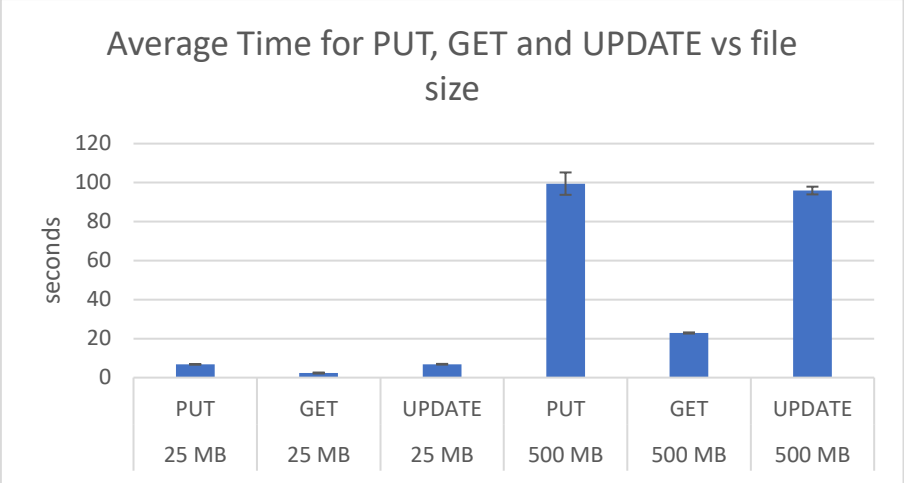
We have utilized the architecture and the membership list from MP2. Each node has the full membership list and it makes the election process and the replication process in case of failures easier. For instance, if the leader node detects 3 simultaneous failures, it initiates the re-replication process. The leader node finds out which files have less than 4 replicas. For each file, it selects the extra replica nodes and asks them to download the file from the current replica nodes. For debugging and performance testing, we successfully utilized MP1 log retriever for fetching the logs and comparing them against the expected log entries. We also debugged the election and replication related bugs using the log retriever implemented in MP1.

PERFORMANCE TESTING

1. **Re-replication time and bandwidth for a 40MB file**
   Since we re-replicate only after 3 failures, we have measured the time and bandwidth after 3 failures in the system.

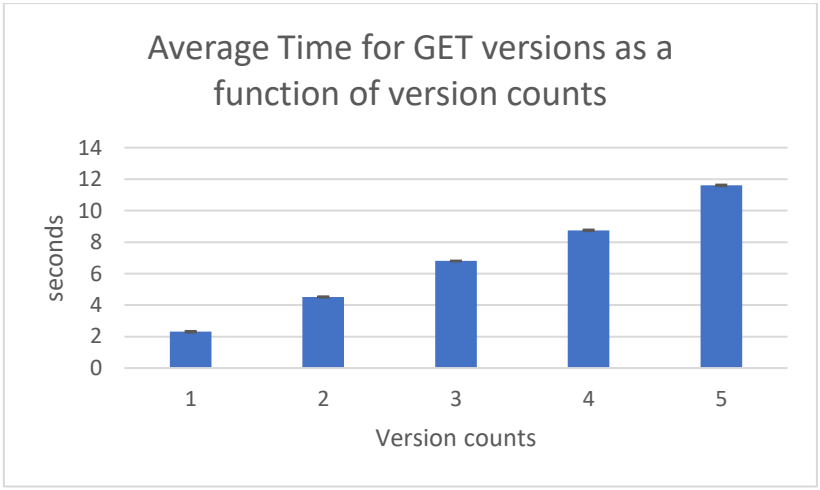


   The average bandwidth is expected as the file is being replicated across 3 nodes at the same and is consistent with the bandwidth usage to replicate one 120MB on a single node.
2. **Times for PUT, GET and UPDATE operations on 25MB and 500MB file**



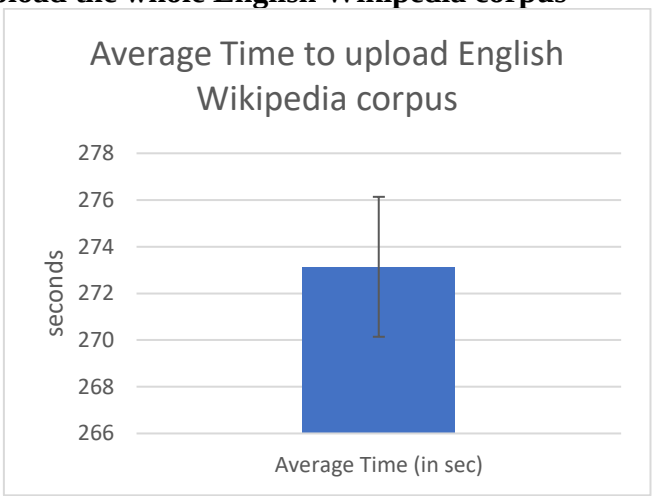
As expected, operations on the 25MB are much faster than on the 500MB file. Furthermore, the GET operation is much faster that PUT and UPDATE as expected. This is because we have set R = 1. Moreover, UPDATE and PUT operations take identical time as expected as UPDATE is essentially a PUT operation.

3. **Time for GET-VERSIONS as function of number of versions**



As expected, the time taken for get-versions operation grows linearly with the number of versions.

4. **Time taken to upload the whole English Wikipedia corpus**



Since the file size is approx. 1.3GB, it takes the most time to upload as expected.