# Design

For file stores, we had a node act as an introducer. This node is chosen by election.

Elections are done with the bully algorithm. Unlike the synchronous case, there could be more than 1 leader elected in a round due to dropped packets, and it is possible that not all nodes reply to a coordinator. We use the usual bully algorithm election timeout to handle these. If more than 1 node claims to be leader and declare itself coordinator, there should be a node that will detect a new leader being claimed too soon and it will force another election. This system will not be truly partition tolerant over extended periods of time. Connections are assumed to be generally but not entirely reliable. File operations are only allowed when 1 leader is decided. The leader will take on the role of the introducer

We have a DNS Daemon process running on VM 1 that acts as the DNS server and stores the IP address of the introducer. The introducer will always use the same port (*INTRODUCER_PORT*). This process is designed to always run on VM1 (under the assumption that VM1 does not terminate). It can be queried and updated by other processes. A lock ensures atomicity.

For membership list handling, we mostly used MP2 but converted it to use TCP sockets in order to use the same socket for both connection messages (LEAVE, JOIN, PING, ACK) as well as file communication messages (GET, PUT, DELETE. LS). Members hip lists are ordered by timestamp joined. Each node treats the two nodes that joined before it as its neighbors. If a node leaves the ring, we update the two previous neighbors to account for this node's leave. This ensures that we can tolerate at least 3 simultaneous failures, since if three simultaneous nodes fail, the neighbor list will be dynamically updated for the next round of PING/ACK and the failures will be detected.

For the filesystem, we had the introducer cache a copy of the map of {file_name: List[file_versions]}. We used a W of min(4, number of nodes) to tolerate 3 simultaneous failures. If there are 3 or fewer nodes, tolerating 3 failures is impossible, so we had as many copies as possible to ensure reliability. We chose 4 for other cases to minimize the number of copies to store. We used R = num nodes. This ensures the node with the latest version is still checked on a read when there are 3 simultaneous failures. The downside of storing fewer copies is more copies are needed on a read.

# Past MP Use

We used MP2 to serve as the baseline for membership in this MP. This provided a foundation for dealing with the filesystem knowing about other process in the system. MP2 hardcoded the introducer, so we had to modify that to support leader election.

We did not find MP1 useful and we generally debugged by testing on localhost with print statements. This was able to help us iterate rapidly and make good progress, though it required some time to port the code over to the VMs leading up to the deadline.

# Measurements

i) Re-replication time and bandwidth upon failure, 40MB file

| | Run 1 | Run 2 | Run 3 | Run 4 | Run 5 | Average |
|---|---|---|---|---|---|---|
| Rereplication Time(s) | 0.189683437 | 0.146691561 | 0.121170282 | 0.126144171 | 0.137574673 | 0.144252825 |
| Bandwidth(kB/s) | 18560 | 18770 | 18210 | 19020 | 17870 | 18486 |

ii) Insert, read, update for 25MB, 500 MB under no failure

25 MB:

| | Run 1 | Run 2 | Run 3 | Run 4 | Run 5 | Average |
|---|---|---|---|---|---|---|
| Insert(s) | 1.868376493 | 1.160346508 | 1.116573572 | 1.714965582 | 1.482499123 | 1.468552256 |
| Read(s) | 1.164074659 | 1.351743221 | 1.515481710 | 1.579698563 | 1.428742886 | 1.407948208 |
| Delete(s) | 1.175193310 | 1.192965984 | 1.078948259 | 1.925763130 | 1.272747993 | 1.329123735 |

These numbers make sense to us – most of the time is us sending the file. The numbers are pretty consistent and in line with our testing.

500MB:

| | Run 1 | Run 2 | Run 3 | Run 4 | Run 5 | Average |
|---|---|---|---|---|---|---|
| Insert | 3.51879 | 3.27264 | 2.48695 | 2.02263 | 2.20764 | 2.70173 |
| Read | 2.73055 | 2.76671 | 2.52872 | 1.99791 | 2.63565 | 2.53191 |
| Delete | 2.90587 | 2.23509 | 1.80143 | 2.05128 | 2.22073 | 2.24288 |

These numbers also make sense to us because most of the time is network latency, and reading the data into the socket.

iii) Time to perform *get-versions*

iv) Time to store entire English Wikipedia Corpus, 4 machines and 8 machines