Kapil Kanwar (kkanwar2), Lawson Probasco (lawsonp2)

CS 425 MP3

Design

Our design uses MP2 internally, so we have one fixed introducer that all other nodes join the cluster with. We also designate a node as the master node in order to simplify a lot of the logic.

Essentially, whenever a client wishes to create / modify / delete a file, it contacts the master node. The master node decides on where the replicas will be by hashing the filename into an index in the membership list, and picking the next 3 members in the ring.

Although we were unable to successfully complete replication under the case of failure, we will describe how we planned on doing it. The master node keeps track of the location of all the files as well as where they are replicated to. All file operations go through the master node, and the master node ultimately decides what each node does with each file. When up to 3 nodes that are not the master node go down, the master node should go through each file, decide the nodes that the file belongs at, and issue commands to a remaining node that still have the file to copy it over. When the master node itself goes down, all I/O gets frozen and elections are held to choose a new master node (described after). When a new master is elected, it waits for some timeout for all nodes to send their list of files to the master node, and same process occurs to re-replicate files as before.

In order to decide who the master node is, we hold elections. Initially, the introducer is the master node. When the current master node fails, all the nodes elect a new master node. At first, we considered just selecting the node with highest ID as the master node, but realized that this would lead to split braining if the membership list was not consistent, and the membership list is not guaranteed to be consistent at any given point in time. So, the nodes perform a fault-tolerant ring election algorithm to select the new master node, which is implemented internally in each node as a state machine.

Debugging

We did not use MP1 very much while debugging this MP, because we created our own ad-hoc testing infrastructure to debug with. Instead of copying the program and running it manually on all the VMs, we created mock networking classes and were able to test all the algorithms locally, which meant that log lines could be emitted directly.

Discussion of Measurements

The below measurements are color-coordinated such that each color corresponds to a specific type of action (red, green, blue corresponds to read, write, update respectively). The file sizes are in GB and the operation latency is measured in milliseconds ms. It’s important to note the bars are in three different clusters where each cluster represents a file size - the bars are simply displaced a little left and right to this size to stack neatly next to one another. The trends shown by these measurements are pretty insightful as to the expense of running our designed

filesystem with multiple machines. As the number of machines increases, the two graphs seemed to show an increase in variation. This is to be expected, as the overhead becomes more costly the more machines we have in the distributed filesystem. What's also very apparent as a trend within each of the plots is that as the size of a file scales up, the overhead becomes less impactful relative to the latency required just to transfer the file. This makes sense because for larger files the time to transfer is much greater than the time to communicate between a few nodes. In terms of raw numbers, 0.5 GB files oftentimes took around 20,000 ms to transfer.