CS 425 - Distributed Systems

Machine Problem 3: Simple Distributed File System

Group 65 : Aruna Parameswaran (netid: aruna2); Sanjit Kumar (netid: sanjitk3)

Design:

The simple distributed file system (SDFS) is designed on top of our distributed group membership protocol from MP2 where each process monitors 3 of its successor processes for failure detection. Here, while implementing the SDFS we have used the same unidirectional ring topology from our MP2 that has features such as versioned file control, leader elections/re-election and data re-replication on failure while taking advantage of the membership protocol and failure detection from the last MP.

The SDFS is designed to have a leader (or coordinator) and multiple non-leader nodes (referred to as regular nodes). The leader node also works as a regular node. The regular nodes all contain a full membership list and know who the leader node is at all times when the system is healthy. Any file system query (get, put, etc) will go through the leader node to the regular nodes that contain the corresponding file being operated on. This node to data (file) mapping is present in the leader node. Basically, it contains metadata about the files and decides location of new writes (which nodes to write to).

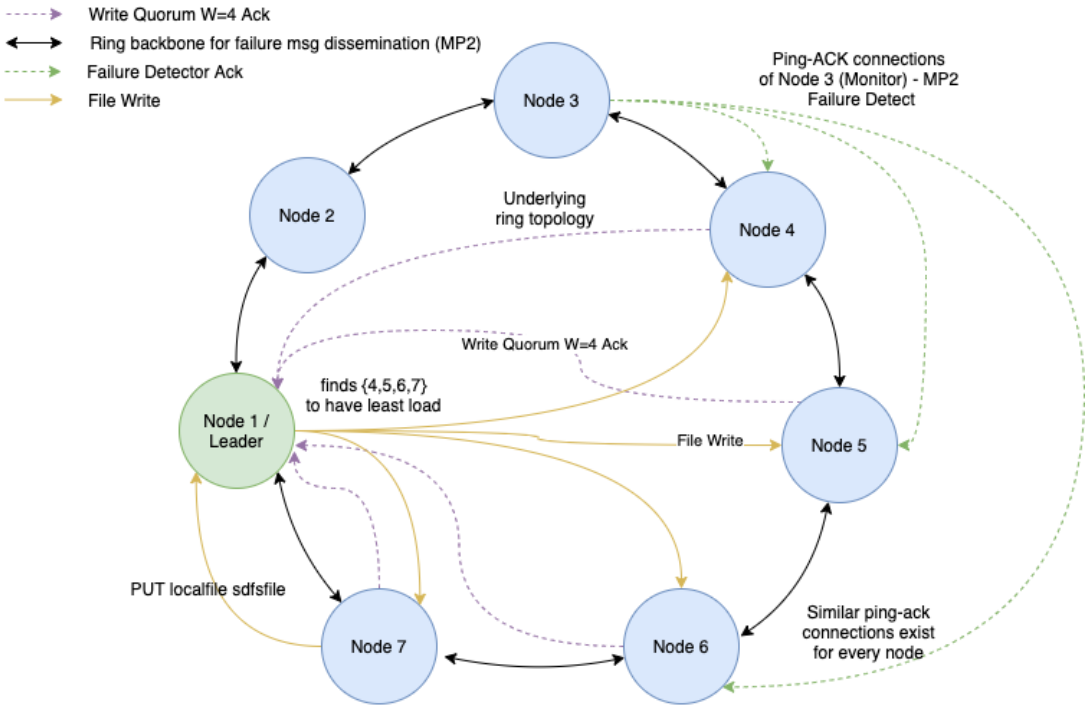
Election happens based on the unique timestamp value, i.e. the VM which joined the earliest will be elected as the leader. Consequently, the neighbors of any leader are guaranteed to win subsequent election rounds, since the addition of nodes in the ring is sequential.

For each file PUT to the system we designed our SDFS to contain 4 replicas (simultaneous failures). Here the replicas are stored in 4 adjacent neighboring nodes in the ring topology. When any node fails, the node that detects its failure disseminates this information. If the failed node is a leader, then its neighbor(s) starts an election round. Once a leader is elected, the leader handles co-ordinating the replication. For any file that is lost on the failed node, the leader computes the sequential next node to store the replica in based on the current list of nodes a file is replicated on. This node is also the common neighbor of all the nodes currently storing the file. In the case there are multiple subsequent failures, this process repeats iteratively until the 4-count replication is satisfied.

The chosen W and R to be W=4/2 +1 = 3 and R=2. This is because in case of 3 simultaneous failures, 2 conflicting writes intersect in at least 1 replicas, hence we go for a quorum majority of replicas. In this case R=2 suffices for there to be an overlap of read and write to maintain consistency of data. Min Value of R+W = 5.

To balance the load, the leader selects the node with the least number of files on it as the target node for storing files to avoid overloading nodes that already have large volumes of data.

While we have not implemented sharding, we split the file into blocks of size of 4K each before transmitting them between nodes (during re-replication and file operations via the leader node) to deal with socket protocol size limitations.



Past MP use

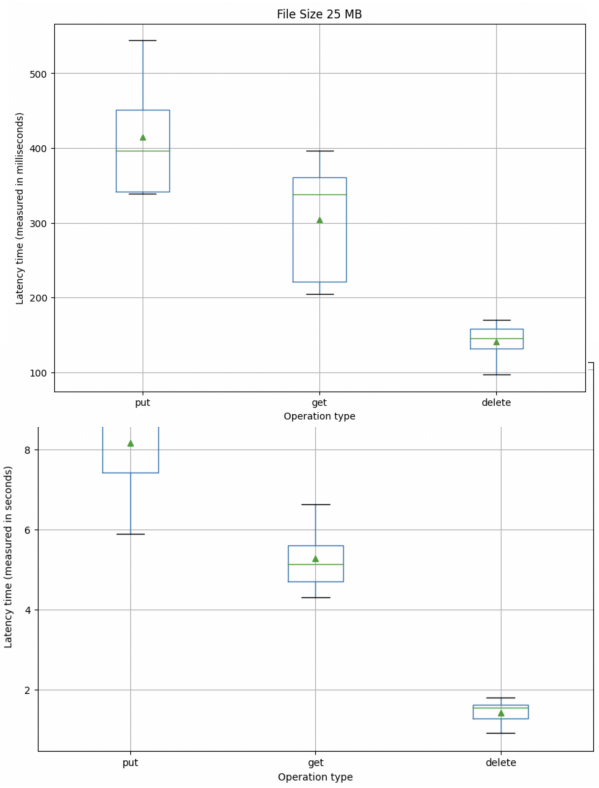
We have used the previous MP2's membership protocol to build the SDFS. The group membership program uses a unidirectional ring topology of the network (for message dissemination) and further the configuration (topology) of monitors for failure detection over the entire network. Each node contains a full membership list but socket connections to only its direct neighbors (ring network topology) but the failures detected are disseminated through the network ring. The leader election protocol runs on top of both the membership list and failure detector code from MP2. The leader's failure is registered through the failure detector. The full membership list information is used to calculate the neighbors for a node during replication. Additionally, we have used MP1 to monitor the file operations and to measure the metrics.

Measurements

(i) Re-replication time and bandwidth used up by failure of a node storing a 40 MB file

With a system containing 8 nodes, the total time taken for the system to converge to a stable state after a failure (non-leader) was measured to be 5.8 seconds. During this period, the system disseminated failure information to all nodes. The leader fetches the file to be replicated and

sends it to the new replica. The overall bandwidth consumed by the network node was found to be approximately 81.36 MB, with an average bandwidth of 14MBps.



(ii) The best case scenario for a CRUD operation is if the querying client happens to be the leader as well. In this case, the leader will directly take the file from its local directory and share to the replica nodes. In the worst case, the querying node first sends the file across to the leader and then the replica nodes, resulting in additional network latency. GET and PUT are nearly comparable in terms of latency, but GET is faster as it only requires 2 ACKS, while PUT needs 3. Delete operation is the fastest, which is expected as bandwidth is not spent transferring data.

(iii) The get-versions command latency and value chosen for num-versions is linearly correlated. As the value for num-version increases, more files are required to be pulled. Additionally, all versions of a file are stored on a single node, hence there is no optimization possible here by parallelizing the get operation.

We tested with a file of size 20 MB.

(iv) The mean time to store the English Wikipedia corpus for 4 machines is around 24 seconds, and for 8 machines is around 27.5 seconds. There's only a slight increase in the latency value, which can be explained by the fact that as the number of nodes increases the communication channel is also more unreliable. The two latencies are comparable, hence showing that SDFS is highly scalable as well.