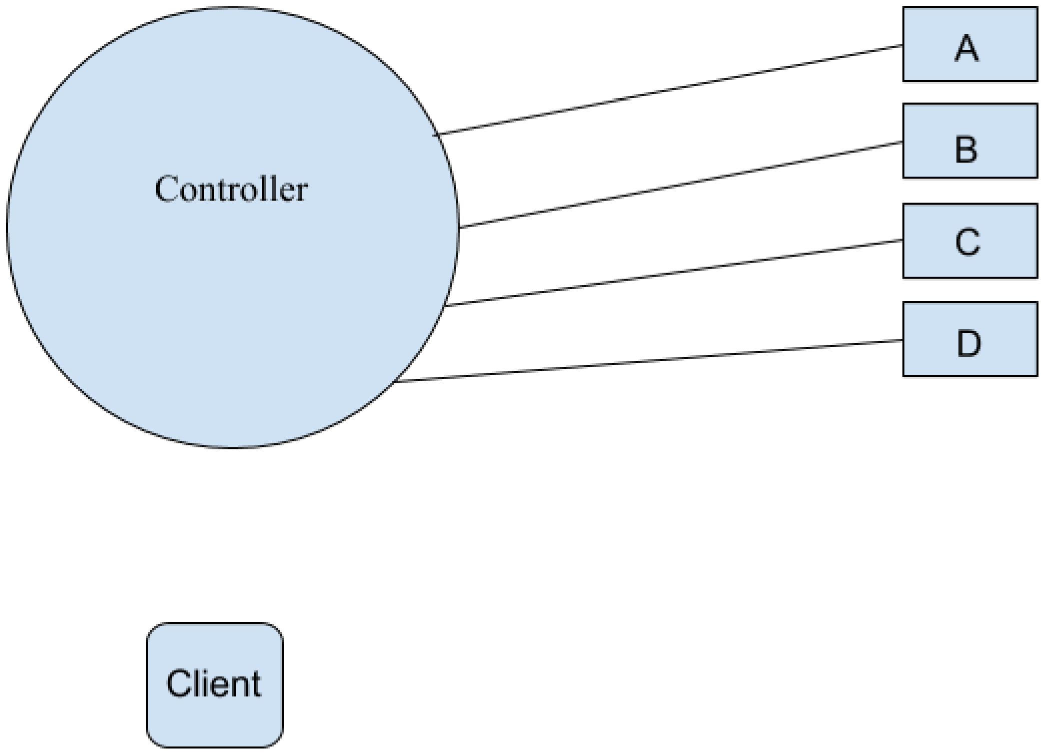
A distributed file system.

Ryan Dielhenn, David Hutchinson.

The distributed file systems contains three main components, all running on seperate servers, the Client, the Storage Nodes, and a Controller. The storage nodes are responsible for storing files from the client, and sending files to the client for retrieval.

The components of the system are laid out below. The lines represent an initial join request of storage nodes A, B, C, and D to the controller. We have a thread on each storage node send subsequent heartbeats with their free disk space. Once the join request completes and the heartbeat thread is started, the nodes begin to listen for incoming messages.



If any request to the controller is made by a client before 3 storage nodes are in the system, it will ignore the request unless it is a node list request.

This is because once we have 3 nodes in the system on the controller we can build a directed cyclic graph of replica assignments for the storage nodes and immediately send those nodes their assignments.

Subsequent join requests will be assigned nodes that have processed the least amount of requests. We do this by keeping all of the nodes in a priority queue on the Controller sorted by requests. A request is counted only when a store request is made, and is also counted for the picked nodes assignments. The picked node is always the one at the top of the queue.

Storage

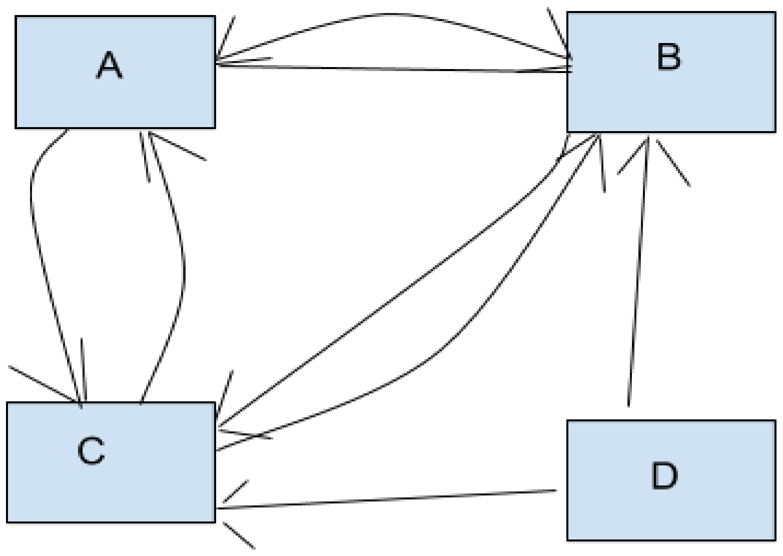
When the client requests to store a file, it contacts the controller with the name of the file. The controller will pick a node from the top of the queue and send client the hostname of the node.

Then client will chunk the file and send it to that node. Our chunk size is 50mb by default but can be configured from client.

Replication is handled when the primary holder receives a store chunk. It sends replicas to its assignments.

Chunks are compressed if they can be reduced in size by 60% or more.

Replication Management



In the case of node failure we use the directed graph to determine dependencies for rereplication. We find all nodes that were replicating to the down node and we send all of them a new assignment.

We pick this assignment assuming all other nodes besides the down node are still active, and each node may get a different assignment depending on who it is already assigned to (to maintain a replication factor of 3 and not re-replicate to the same node twice).

So, if B went down D would re-replicate its primary data to A, and C would send its primary data to D. Then either A or C would be picked to merge their primaries with replicas of Cs data. We merge the bloom filter of the down node into the new primary holders so that all subsequent requests for that data is sent to the new primary holder.

The replication factor is maintained by the new primary holder on the storage node side. If the new primary holder is assigned to one of the down node’s old assignments, we just alert that node that ownership of the replicas has changed. If an assignment of the new primary holder was not one of the down nodes assignments, then the new primary node will re-replicate all of the down nodes data to that assignment and claim ownership of it.

Retrieval

When client requests to retrieve a file, controller checks the filename against every node’s bloom filter and sends client a list of possible nodes that have the file. Then client requests chunks of the file from that node. Upon retrieval, primary data holders will check the previously saved valid checksum against the chunks current checksum. If they do not match requests are pipelined through replica assignments to heal the data.

Retrospective

Currently, we put all chunks of a file on one node but we could easily spread chunks of a file across all nodes by putting chunk names in bloom filters instead of filenames, and have nodes claim owner-ship over chunks instead of files for handling re-replication in case of node-failure. This way we could multi-thread retrieval quite easily with a random access file. We also do not handle concurrent node failures.