MP2: Simple Distributed File System Report

Alvina Waseem (awaseem2), Kartikeya Sharma (ksharma), James Timotiwu (jit2)

Design

The SDFS has exactly one master node at any given point of time. The master responds to every CLI command executed from any machine. It supports five different networked operations: `put`, `get`, `delete`, `ls`, `store`. Each machine acts as a server and client. When a client issues a `put`, the master chooses a random set of IPs up to the replication factor and returns to the client. The client will then directly upload its file to all machines in the set before completing his put.

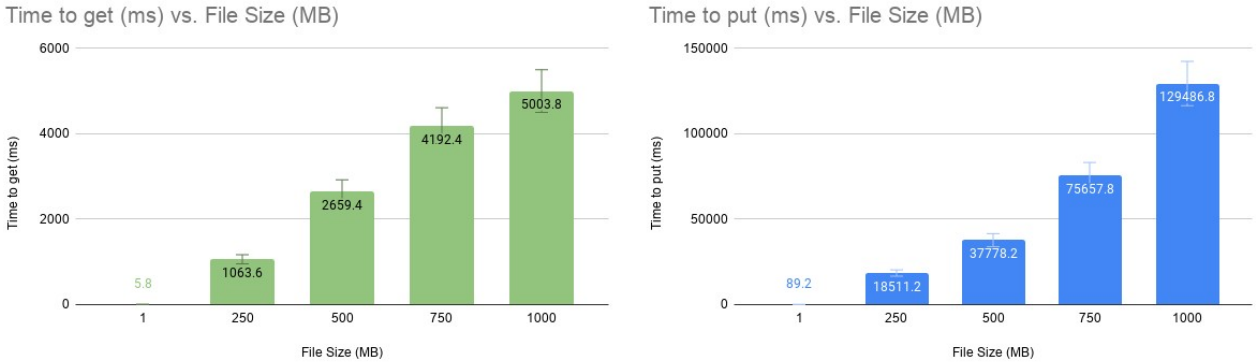
Masters are also responsible for bookkeeping file replica information so nodes who make `get` requests can be routed to the desired files. When a `get` request is issued, the master replies with a list of network locations/IP addresses. The client receives the list and randomly chooses a node, to which it directly connects to the node holding the desired replica.

The master runs a lock service to ensure concurrency is properly handled. Before any put/get operation, the client requests a lock to acquire from the master. Every file in the SDFS has its own special lock, so multiple files can be read and written simultaneously from any node while maintaining consistency guarantees. Each lock request is assigned a session ID. A session ID is used to keep track of how long a lock had been used, or if the lock for that particular session had been released. This ensures locks will not permanently deadlock, and cannot be released upon master failure.

The bully algorithm is implemented for leader election. A node can call for an election when it detects its master has failed. When a node calls for election, it sends a message to nodes with higher IDs than itself whether or not they are alive. If at least one node with a higher ID replies, the node will receive an `OkMsg` from the higher ID node and end its candidacy. The node that sent the acknowledgement will run its own election, sending an `ElectionMsg` to nodes with higher IDs. The node who never receives an acknowledgement from a higher ID node will send a `CoordinatorMsg` to all nodes with lower IDs that it has declared itself master. All nodes receiving the `CoordinatorMsg` will send a list of all files each one is holding to the master, and the master will use the data to rebuild the original state of the failed master.

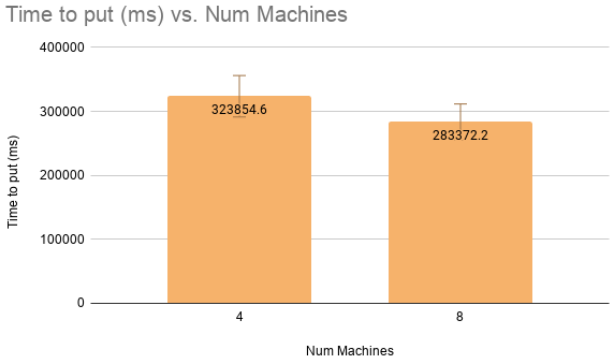
SDFS is tolerant to three machine failures. When a file is first requested to be uploaded, the master ensures that it is uploaded to four nodes. Whenever the current master detects a failure of a node, it goes over a map containing all the files stored by that node. For each such file, it requests an alive node containing the file to upload it to another node which doesn't contain the file - this is retried until either all replicas are dead (which can't happen since we're limited to three failures), or no alive node is available to create an additional replica (which won't happen once we have at least seven machines, in case of three failures). In this way, it is always ensured that four replicas of a file are always alive.

Performance Evaluation



1

Figure 1 above shows a positive correlation between the time taken in milliseconds to get or put a file versus the size of the files in megabytes. This is expected behavior because networks are limited by the packet transmission rates and bandwidth, whether or not buffers for routing links between two nodes exist. A distributed file system must transmit its files to multiple storage locations in their cluster, so we must consider factors such as added I/O overhead to read the file from disk and prepare the contents of the file for transmission.



2

Unsurprisingly, the time to put files into the distributed file system trend somewhat flat, with a non to negative correlation if we compare between four or eight machines present in the group. The difference is negligible because we also see that error bars between the two setups overlap. This is likely because the files are still being sent to the same number of machines, despite a higher number in the group. Otherwise, the negative correlation could suggest that having more machines might help evenly distribute the load of where to create replicas. Further, some nodes may be more local or have more free bandwidth between the networks, and more machines would potentially increase that probability.