Cody Guldner - cguldne2
Joseph Zabinski - jzabins2

ECE 428 MP3 Report

We used a chord-like approach in our design of a distributed file system. We hashed the sdfs file name in order to map the file to one of our servers. The file is stored on the server it maps to and the next three servers in the ring. Each client uses the hash function to figure out where a particular file is stored. As a result, clients communicate directly with servers, and our design does not have a centralized server. Furthermore, the hash function helps to distribute the files evenly across the servers.

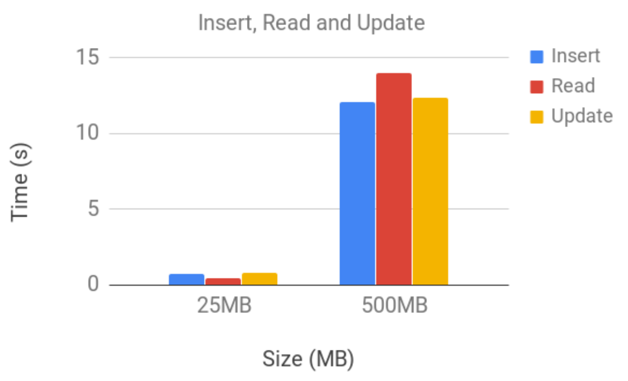
Since we can have at most three simultaneous failures, we decided to keep four replicas of each file. This ensures that we have at least one copy of each file even when there are three failures. When there is a failure, each server checks if any of its files were stored on the failed server. If a functioning server has a file that was stored on a failed server, the functioning server sends the file to another functioning server to maintain four copies of the file. Similarly, when a new server joins, other servers use the hash function to check which files the new server should have, and they send the new server those files.

We decided to write to all (4) and read from one, which ensures read/write consistency. Since writes go to all servers (with replicas), each server has all version of a file. This makes it simple to totally order files in the versioned file system. However, writes are slower because all four servers must provide an ack for the write to succeed. In contrast, reads and get-versions are fast (and simple) because each server has every version, so only one server must respond to the request.

We didn’t use MP1 very much in our development. MP1 was useful at first; however, we found that ls and store were much easier to work with than grep, so we used ls and store for debugging once we implemented those functions.

We measured the replication time when one server with a 38.1MB file fails. With five trials, the time for replication had a mean of 2.273 seconds with a standard deviation of 0.432 seconds. This was longer than we expected because a ~40MB file can be transferred very quickly. We think that replication was taking a long time because due to the relatively long timeout period as well as the relative infrequency of pings in our design.

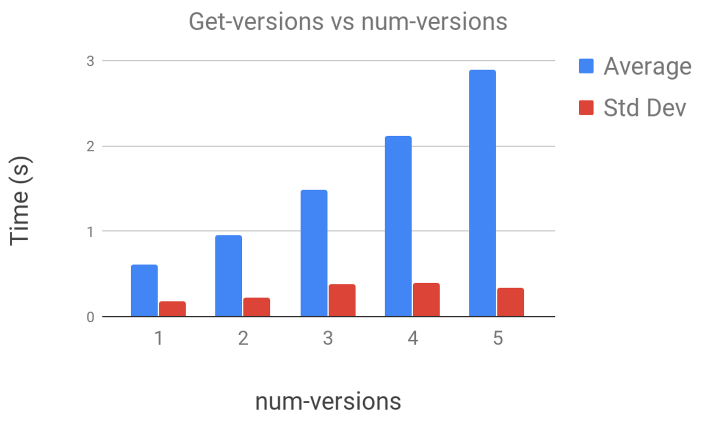
We measured the bandwidth usage to be 34.479 megabytes per second when a server fails. We expected this number to be much higher because the file is sent over the network three times when a server dies.The file is sent three times because each surviving server (with the file) sends a copy of the file to a new server. For a ~40MB file, this



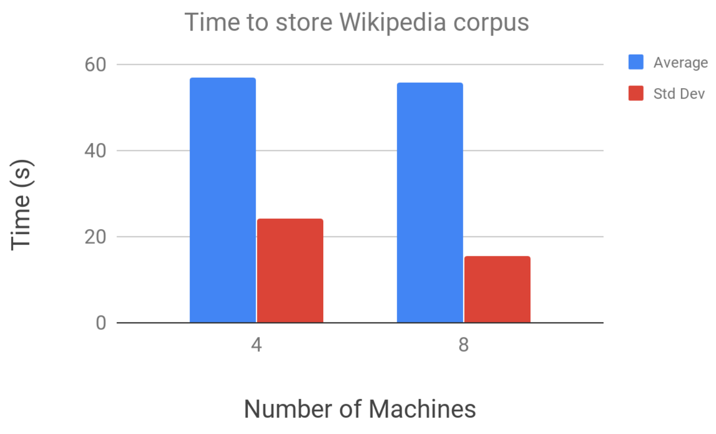
would result in ~120MB of data being transferred. One possible reason for a low bandwidth measurement could be that tcpdump was averaging the total amount of bytes sent over a couple of seconds. This could result in a low average bandwidth, even if the peak bandwidth was high.

We found that the times for inserting files were surprisingly fast and fairly operation agnostic. We expected insert and update to take much more time than read, because we are using W=4 and R=1 but found that they are about the same. This could be because the VMs operate at similar speeds and the network is very good. It does makes sense that insert and update would be about the same time, because update is just inserting the file, and renaming the other files with the same name.

The dependence between the time to return on `get-versions` and the number of versions was exactly as we were expecting. There is a linear dependence between the time and the number of versions because the amount of data transferred increased linearly. We used a randomly generated 25MB file while measuring the time, and even with 5 versions, it was fairly fast, with less than 3 seconds return time.



It took on average around 60 seconds to store ~1.1GB file into our SDFS. We think that this is a reasonable amount of time, as data transfer tends to be very slow. We were expecting the time for 4 and 8 machines to be exactly the same, because we are only replicating to 4 machines total. However, there is a large amount of variance in our data, and we couldn't get very consistent results.


Time to store Wikipedia corpus
Average
Std Dev
Time (s)
Number of Machines