Distributed System MP3: Simple Distributed File System

Group 41 Cheng-Hsuan Huang(chhuang5) Yen-Chen Yeh(yy63)

Design

We implemented a C++-based Simple Distributed File System (SDFS) built on top of MP2 (Membership Protocol). We introduce a leader into the membership list, responsible for making decisions regarding the version numbers and replica servers. Additionally, we designed a Failure Recovery system capable of tolerating failures.

Leader election

We assume that every member sees the same membership list and chooses the server with the smallest ID as the leader. If the leader fails, the remaining servers will become aware of it and select a new leader.

Filelist

We designed a 'filelist' class that stores information about SDFS files, which includes both global and local values. The global values are piggybacked with gossip messages used in MP2, while the local values are stored locally.

Global value:

1. FileMetadataMap: an unordered map where the keys represent SDFS file names, and the values include their versions and the IDs of the replica servers that store the files.
2. VersionNumber: This number can only be increased by the leader when receiving 'put,' 'get,' or 'delete' commands. It ensures that multiple actions can be performed on the same SDFS file.

Local value:

1. FileMap: This map stores information about the replicas and their versions stored on the local server. It is checked when the leader requests information about a specific file.

Put / Get flow

For the 'put' command, a client will send a request to its server, and the server will contact the leader to obtain information about the replica servers that store the SDFS file. The leader will check if the SDFS file exists. If it doesn't, the leader will generate random replica server IDs and return them to the client. If the file does exist, the leader will send back the IDs of the replica servers that store the SDFS file. The server will then begin sending the file to those replica servers. A similar flow is observed for the 'get' command. The leader will return the IDs of the replica servers if the SDFS file exists, or it will respond with 'not found.' When the server receives the IDs of the replica servers, it will request their versions and ask the replica server with the highest version for the file.

Number of Replica is 4, it can tolerate at most three servers fail simultaneously and enable immediately get command. *Quorum sizes*, R (for 'put') and W (for 'get'), are both set to 3, and having R + W greater than the number of replicas ensures that there will always be at least one replica common to both 'get' and 'put' operations.

Multiple get/put

The leader maintains a command queue that handles continuous 'get' and 'put' commands. If there are four successive identical commands, the leader will reorder the queue to meet the specified criteria. This also ensures that no 'get' and 'put' commands are executed on the same SDFS file.

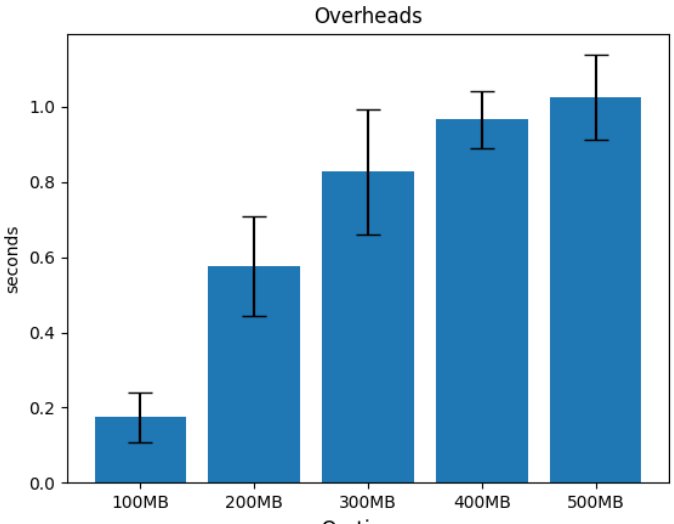
Failure Recovery System

All the servers can detect failures and add the failed servers to the list of failed servers. If the leader identifies that some servers have failed, it will collect information about the replicas stored on those failed servers and provide a list of new replica servers to the non-failed servers that also have those replicas. Subsequently, the servers containing the replicas will begin transferring the SDFS files to the new replica servers. After the recovery process is complete, the leader will notify all remaining servers that the previously failed server has been successfully recovered and remove it from the list of failed servers.

Past MP Use

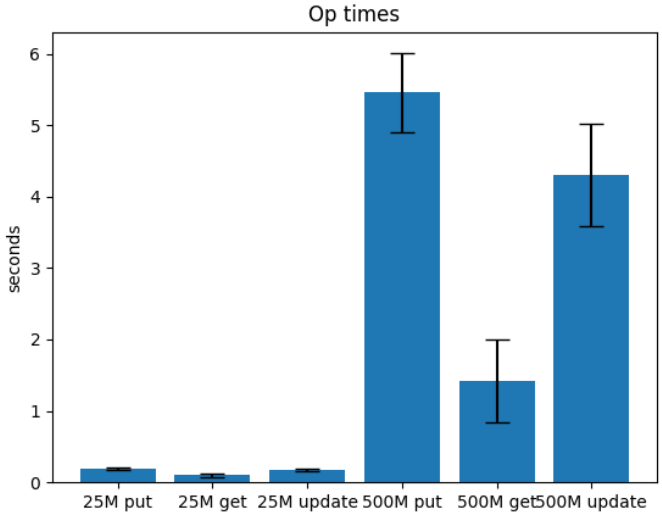
We use the error detection from MP2 to detect if a failure occurred in the system and remove all its replicas from the global list and begin transferring the SDFS files to a new replica server.
The leader server sends out the latest version of the global file list via piggybacking the gossip messages from MP2.
MP1 is mainly for debugging issues. When we finish an action, we put it down on a local log file, which we can later use the grep function from MP1 to know what a specific server has done.

Measurements



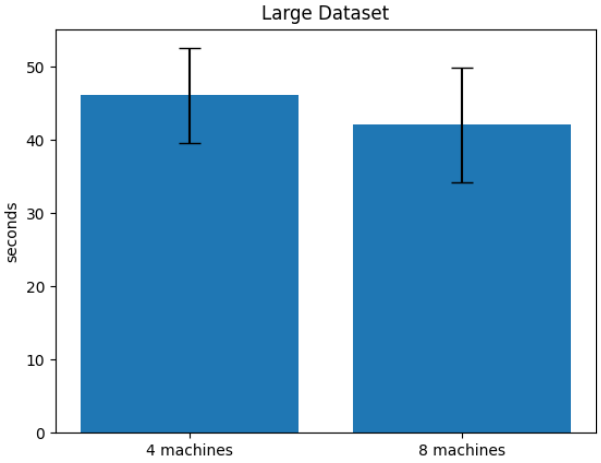
(i) Overheads

When a failure occurs, the leader server finds out all the SDFS files that the failed server contains. After that, he asks one of the servers with the same replicas to replicate the file to another not-yet-replicated server. Therefore, it makes sense that the average time increases as the file size increases.



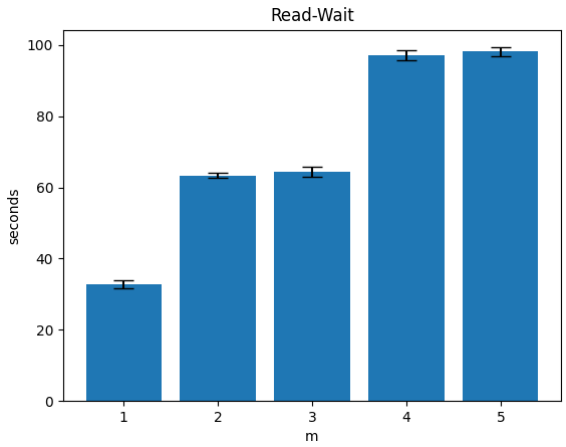
(ii) Op times

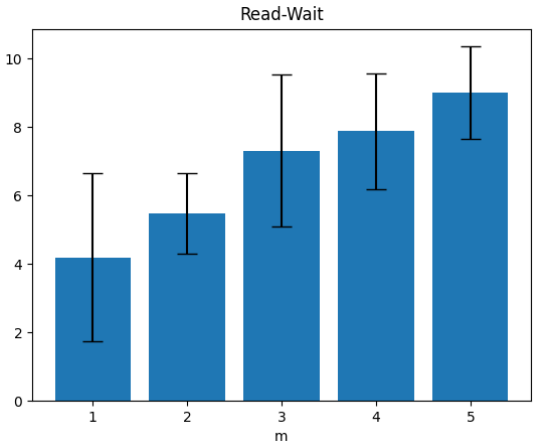
When a server writes a file, it gets replicated to 4 servers, so the bigger the file is, the longer it is going to take to write a file.
For read, the leader only goes to 3 replica servers and find the one with the latest version and return it to the query server, so it takes significantly less time than read.
For update, we only write to 3 of the replica servers, so the time would also be less than write.

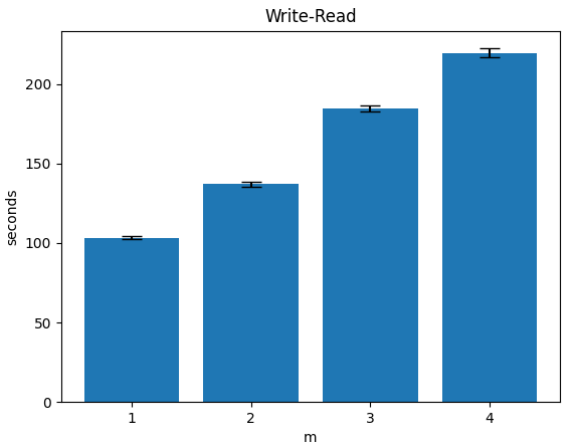


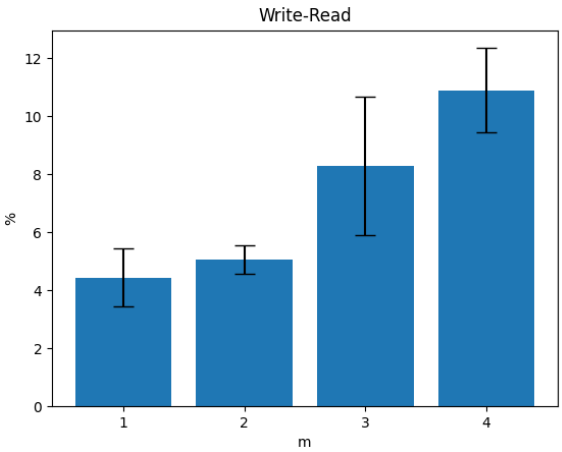
(iii) Large Dataset

The result should be nearly identical because there are always 4 replicas needed to write a file.









(iv) Read-Wait (file size: 6GB, typical read time ~30s)

A read command can run two at a time. Therefore, in an optimal case, two would come in at a time, then they finish at around the same time, then two more come in in their place…

For the second graph, we think the overhead might come from the leader asking the replicas their versions. That overhead would accumulate through the number of reads; therefore, the overhead percentage would become higher and higher.

(v) Write-Read

A write command is exclusive, so it can only run by itself. However, for m=4, the last command to finish is a write command because of the starvation rule.

Same as (iv), we think the reason for graph 2 is the same. However, the overhead for write is bigger because it has to actually replicate the files to different replicas.