CS425 MP3 Report

Group Members: Sarthak Chakraborty, Raunak Shah

Design

In this system, the machine with the highest VM ID is designated as the leader, elected periodically every 5 seconds. The system maintains two key components: a file replication dictionary mapping file names to the machines storing their replicas, and a list of failed nodes. A replication factor of 4 is used, ensuring at least one replica is alive even if 3 failures occur. The system uses a WRITE_QUORUM of 4 and a READ_QUORUM of 1. All client operations are sent to the leader and added to write and read queues, with requests processed in the order they're received.

To handle starvation: We keep a counter for operations in both queues, starting at 0. The counter increases for all writes when a read is dequeued, and for all reads when a write is dequeued. An operation is dequeued when its counter reaches 4, ensuring no read waits for more than 4 writes, and vice versa.

To handle concurrent operations: We maintain a write lock set and a read lock dictionary. We add a file to the write lock set when the put begins, and remove it when the put ends. No other writes or reads to the file are allowed in between. Similarly, every time a read operation to a file begins, we increment read_lock_dict[file]. When reading is complete, we decrement read_lock_dict[file]. We make sure read_lock_dict[file] is always <= 2, i.e. no other process can read the file once the lock equals 2. These locks ensure that concurrent write-write and write-read operations to the same file occur sequentially, and no more than 2 reads can occur to the same file at once.

When the leader processes a get (put) request, it sends the client machines with the relevant replicas. The client then writes (reads) the data to (from) the replicas based on the write (read) quorum. When the operation is complete an ACK is sent to the leader and the client. When all writes are complete, the leader updates the file replication dictionary. Deletes follow a similar set of steps as put operations.

Failures: When a VM fails, the leader machine initiates a re-replication process. It identifies the replicas (e.g., X) in the failed machine using the file replication dictionary. The leader then instructs a non-failed machine (e.g., Y) that holds the same replica to replicate it onto a new VM (e.g., Z) via a socket connection. After successful re-replication, the file replication dictionary is updated. This process continues until all files from the failed machine are re-replicated to maintain the total number of replicas equal to the replication factor. Once this is achieved, the failed node is removed from the failed nodes list.

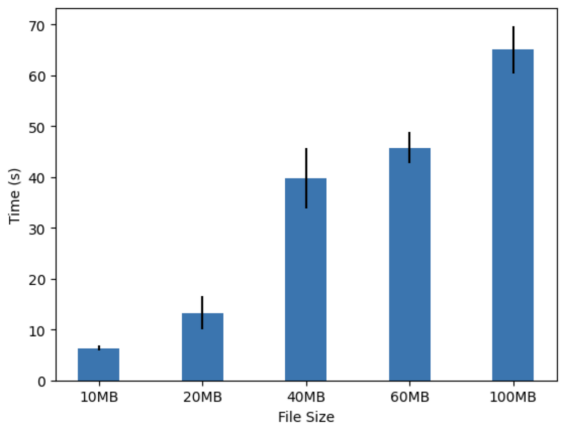
Past MP Use:

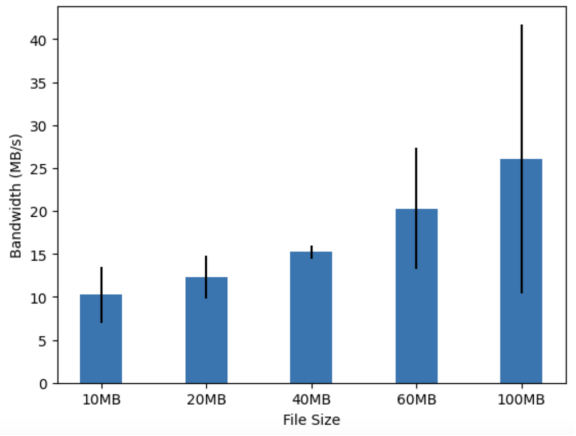
We used the failure detection class from MP2 as well as the membership list (including functions for updating and deleting members from it). We used MP1 to help debug logs that we stored on each machine.

Measurements:

(i) Re-replication Overheads

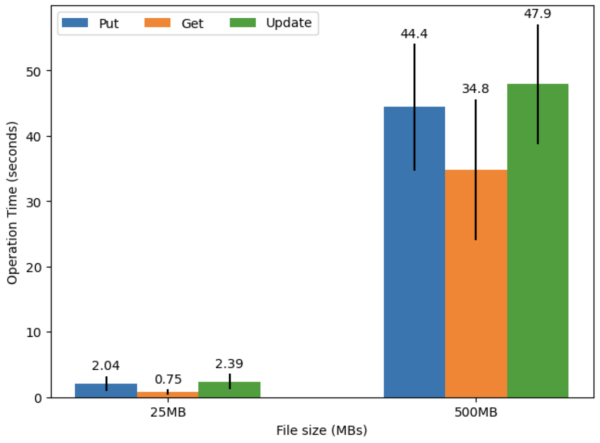
Socket Buffer Size = 4KB





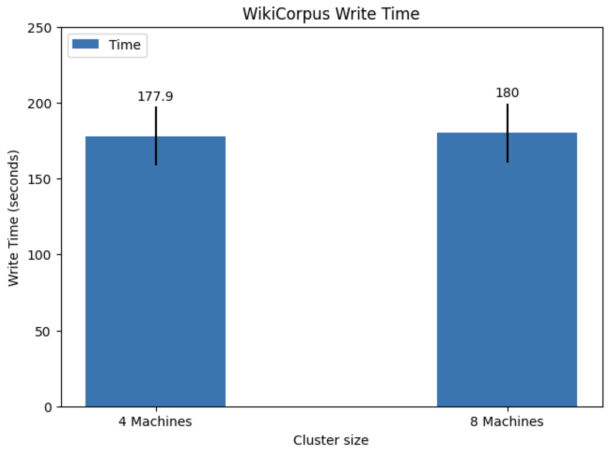
The re-replication time increases as the file size increases, since the amount of data to write to disk increases as well. Similarly the bandwidth also increases, but at a slower rate compared to the re-replication time.

(ii) Op times



Socket Buffer size = 32MB. The time to insert a 500MB file is significantly greater than 25MBs. The relationship is almost proportional to the file size. Additionally, get operations typically finish slightly before insert and update operations.

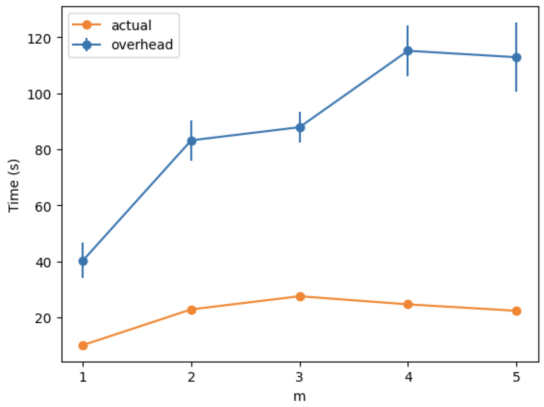
(iii) Large dataset



We insert the compressed wikicorpus into the sdfs. The time of insertion is similar for both the 4 machine case and 8 machine case, which makes sense since our replication factor is = 4. Increasing the number of machines is unlikely to make a difference.

(iv) Read-Wait

We use a 300MB file which typically takes (36.63 ± 6.403)s to read for buffer size 128KB.



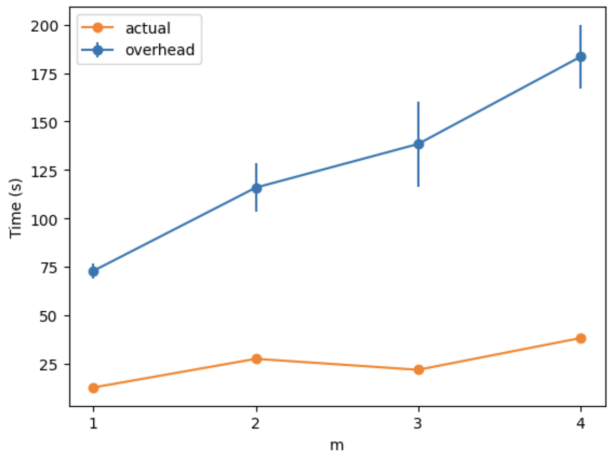
With increasing the value of m, the read latency of the last reader increases in general. However, since 2 reads can be done to a file concurrently, the latency value remains the same for 2 consecutive 'm' values (For example, m=2 & 3 and m=4 & 5).

Ideal (best case) read time = R (taking min time=30.2s), Overhead = T - P

m	1	2	3	4	5
Ideal Time P	R=30.2	2R=60.4	2R=60.4	3R=90.2	3R=90.2
Overheads	10.03	22.8	27.5	24.6	22.3

(v) Write-Read

500MB File || Socket buffer 32MB || Typical Write Time = (35.125 ± 6.915)s || Typical Read Time = (10.228 ± 6.404)s



We notice that the time to complete the last request linearly increases with m. This is expected since all the operations considered are conflict operations (write-write or write-read). There should be an order among the conflicting operations and they are not allowed to execute concurrently. Hence the trend linearly increases since every operation is executed sequentially.

Ideal Time until last operation occurs (vs m) - taking W = 35.1-6.9 = 28.2 and R=10.2-6.4=3.8

m	1	2	3	4
Ideal Time P	2W+R=60.2	3W+R=88.4	4W+R=116.6	5W+R=144.8
Overheads	12.64	27.45	21.8	38.2