CS425 MP3 Report

Yuechuan Zhang (yz134) Xinshuo Lei (xinshuo3)

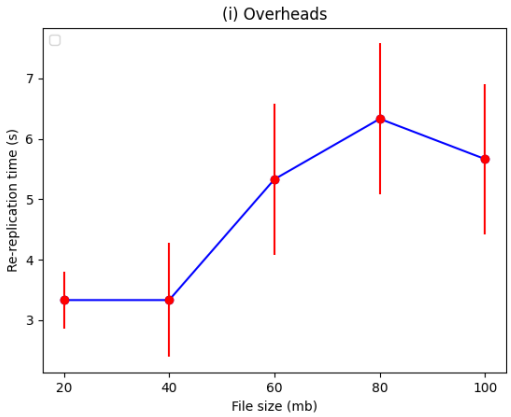
Design

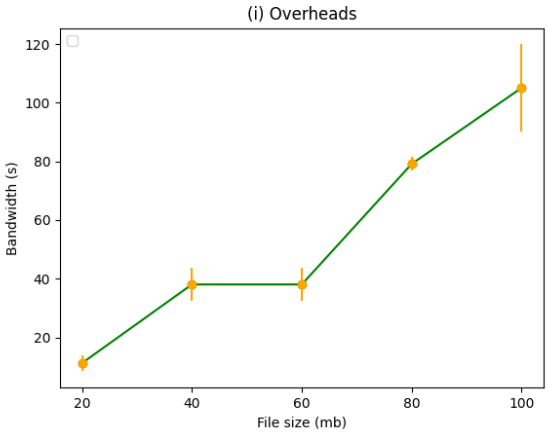
Our SDFS design includes four major components.

1) Leader election service: we implemented a quorum-based leader election service that uses the membership service in MP2 for multicast and detecting leader failure. At each round, each participant multicast an election request and then votes for the smallest node ID they have observed before a vote timeout. The node receiving majority vote becomes the leader and multicasts leader notification. In case of leader failure or higher round ID messages (possibly due to timeouts), re-election occurs.
2) File metadata service: this service is hosted only by an elected leader and provides SDFS clients and file servers with a consistent view of the current file distribution. The service periodically collects file metadata from file servers, decides on a new file distribution view to meet replication requirements (4 replicas in the case of MP3), and informs file servers of its decision. The service also dictates the allocation of new replicas when PUT requests are made by a SDFS client.
3) File master: file master is appointed by the central file metadata service and is unique for each file. Its main responsibility is to schedule SDFS client requests. Specifically, it manages two queues: a main queue and a write queue. All requests are added to the main queue in the order they are received. Write requests are added to both queues. Requests are executed according to their position in the main queue with two exceptions:
   i. If the current number of reader (global variable) < 2 and there is no writer on the file, the read request is immediately removed from the queue and executed
   ii. When a read is executed, the waiting round variable for each write request in the write queue is incremented by one. When the wait round for the head of the write queue reaches 4, the write request is scheduled to be executed next, ignoring all requests before it in the queue, which prevents starvation for write. This design also prevents starvation for file read in a trivial way. No write requests that are made after a read request can be executed before that read request.
4) File servant: This service is hosted on each machine and handles file operations. It reports metadata to the central file metadata service and also complies with metadata change ruled by the file metadata service. When a file master is lost. A file servant with the latest file version id is appointed by the metadata service as the new master.

Past MP use: the distributed log collector remains available and is used for debugging in MP3.

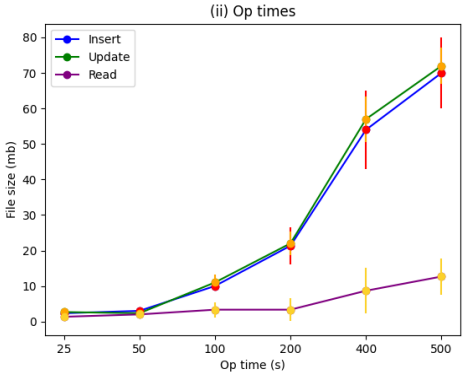
(i)



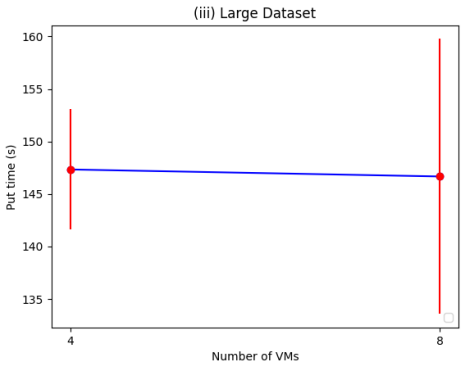


We tested the overhead of a failure on 10 vms. The re-replication times only vary by a couple of seconds because the time it takes to send files that are less than 100MB over TCP differs very little. The bandwidth increases with file size, which is expected.

(ii)



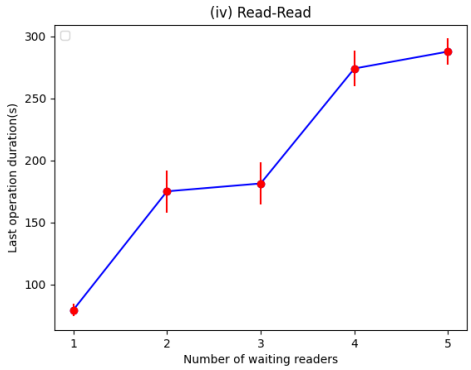
We observe that the insert and update operation times are approximately the same. This makes sense since when there are no other pending requests, an update is basically an insert and needs to write to all replicas. Read time is significantly less than insert and update time. This is due to the fact that only one copy of the data needs to be sent over the network, while insert and update operate on all replicas.

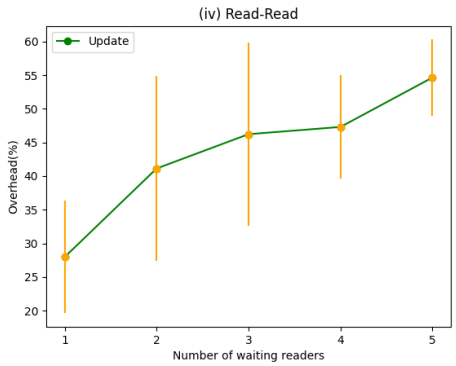


(iii) Large Dataset (graph on the right)

The average put time of English wikipedia corpus (compressed .tgz) is roughly the same for 4 VMs and 8 VMs. This is expected because the put time depends on the replication factor, and the replication factor is set to 4 in both cases. Since we are using TCP to send files, the standard deviation is relatively large.

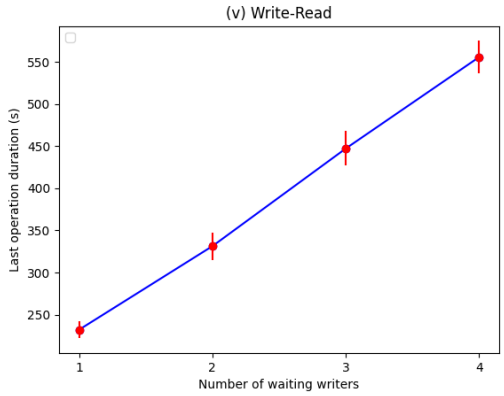
(iv):

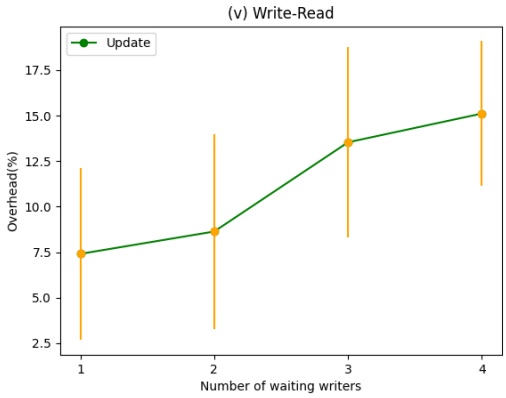




We didn't do well on load balancing, and the file master is responsible for sending replicas for all read requests. This introduces an extra overhead due to bandwidth constraints. We observe that the operation time for 2 and 3, also 4 and 5 waiting readers are approximately the same. This is due to the fact that we allow 2 simultaneous reads.

(v):





The last operation time increases at a steady rate. This is due to write-after-write conflict. Thus, once a write starts, another write cannot start until it finishes. Same with read-write. The overhead increase is mainly due to the cost of managing the queue and maintaining a comprehensive metadata at the file metadata service.