MP3 Report

Group 52: Shashwat Jaiswal (sj74) and Xinying Zheng (xz112)

Design: SDFS is implemented using Java v11. It uses gRPC v1.58.0 for remote execution of its services and Protocol Buffers (ProtoBuf) for data serialization and deserialization. It uses HTTP 2.0 as its underlying protocol and supports parallel connections. However, we limit the usage of gRPC to the services that require reliable communication like introductions and commands. We implement heartbeats using UDP but couple it with Google's Protocol Buffers for ease of management. We leverage Apache Maven as the package manager for Java for easy cross-platform deployments.

SDFS comprises three services, namely, CoreService, SDFSNodeService and SDFSInternalService, and a client library. These are described below:

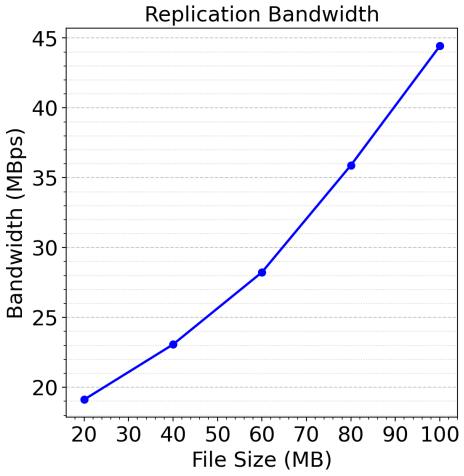
1. CoreService manages critical system components like gossiping, heartbeats, suspicion mechanism, leader election and related services responsible for the overall coordination of the cluster. This also includes membership exchanges and failure management and recovery, granular read and write locks and starvation management.
2. SDFSNodeService hosts services and RPCs that face the client. This includes the put, get and delete operations required by the MP specification document along with a few other other APIs like ls and store. Lastly, this service also exposes a replicateShard service that is used by the leader for failure recovery and management.
3. SDFSInternalService is a private service that interacts directly with the filesystem of the VMs on which SDFS is running. This exposes underlying SDFS RPCs that perform granular filesystem operations for specific shards.
4. The SDFS Client Library contains critical client-side logic for seamless interaction with SDFS cluster services. It involves operations such as get, put and delete apart from ls and store as per the MP specification. It is easy to use and comes with a wide range of command line options and arguments to support diverse use cases as per client requirements.

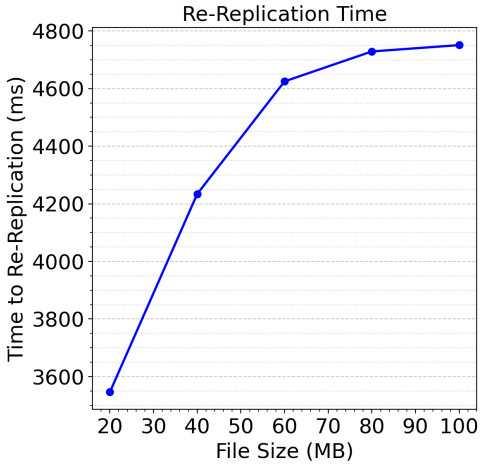
SDFS uses a replication factor of 4 in order to meet the MP specification criteria of 3 concurrent failures. Moreover, we use a custom fairness policy in the java.util.concurrent package's ReentrantReadWriteLock to avoid starvation. This ensures that our requests are given access to the critical section in the first come first serve order while also ensuring the writes do not wait for more than 3 reads.

Past MP Use: We built our system on top of our MP2 submission without any modifications in the latter. We use the same membership lists generated by the heartbeating and failure detection service we implemented in MP2 in order to assign replica locations and manage other entities of SDFS. We also tried using MP1 log querier for debugging purposes but eventually resorted to conventional style brute force debugging since it wasn't very difficult to do so.

Measurements:

I. Overheads: Re-replication times given below include the time to detect failure (which is 2.5s in theory) along with the time to re-replicate. The following measurements are made upon 1 node failure only. We take the mean of 6 observations for each measurement. The re-replication time converges at high values due to the shard size being 64MB, i.e., when we replicate a file (> 64MB) in size, the system tries to parallely replicate the shards which have a maximum size of 64 MB, thereby converging the max time needed. The bandwidth of the system for replication is calculated "excluding" the time to detect failure. We see an increasing trend as we use smaller files. However, it appears to converge at around 60MBps (not shown but evident below).



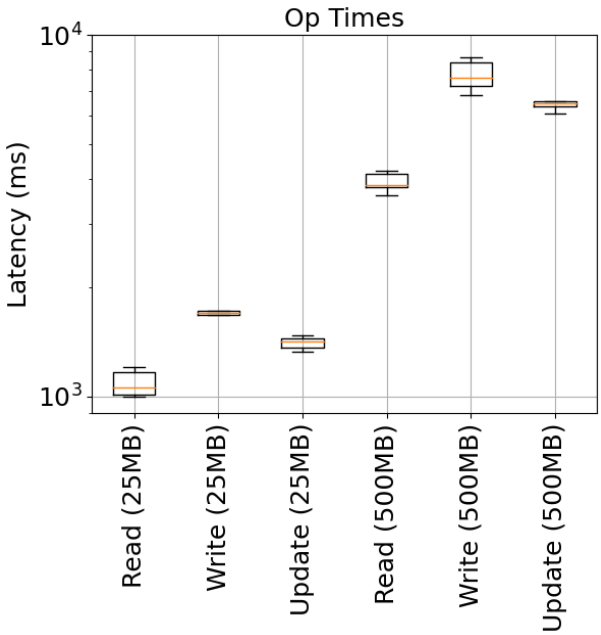


We use the following file sizes - 20 MB, 40 MB, 60 MB, 80 MB and 100 MB.

II. Op Time (ms) - We again take 6 measurements per operation per file size which are given below.

A. 25 MB
   1. Put: 1723, 1675, 1678, 1701, 1800, 1709
   2. Update: 1478, 1426, 1454, 1327, 1401, 1350
   3. Get: 1194, 1095, 1002, 1201, 998, 1024

B. 500 MB
   1. Put: 8507, 6832, 7240, 8674, 7992, 7265
   2. Update: 6574, 6303, 6053, 6448, 7103, 6509
   3. Get: 3794, 5481, 3871, 3603, 3798, 4201

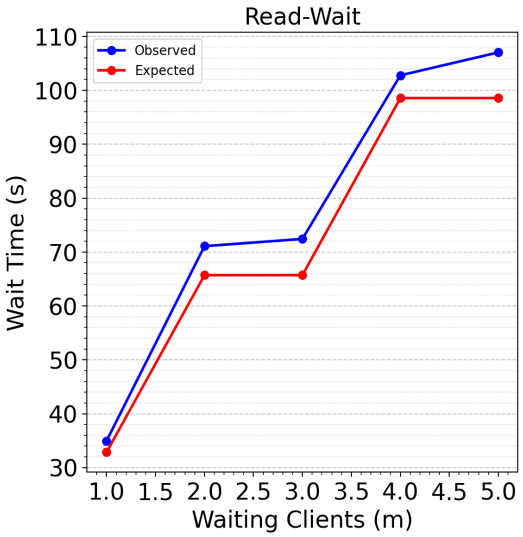


III. Large datasets

The Wikipedia text given is approximately 1.3 GB in size. To write it into SDFS, it took around 27945ms for 4 machines and 20133ms for 8 machines. These numbers are the means of 6 observations. We can easily explain this as for when the system uses only 4 machines, the writes per machine is double as when the system is running on 8 machines. However, we are able to mitigate this considerably due to multithreaded writes (in SDFS).

IV. Read-wait

- File size: 4GB, No. of VMs: 10, Normal read time: 32854ms
- Idea Time (ms) : (ceil((m+1)/2)*Actual Time)
- Discussion: The file we chose here is 2 GB, and the average reading time is around 36s. We find that generally, our system is around 20~30% slower than the theoretical ideal time.



V. Write-read

- File size: 2GB, No. of VMs: 10, Normal write time: 45687ms, Normal read time: 36854ms
- Discussion: The file is 2GB, the average writing time is around 45s and the reading time is 30s. Since write-read and write-write are conflict operations, each operation cannot be started until the last operation is finished. So the operations are performed in a sequential order.