CS 425 MP3

Jiayi Luo (jiayil5)
Zhoutong Jiang (zjiang30)

For this MP, we build our model based on UDP (for failure detection) and RPC (for file transfer). The failure detection (which is based on SWIM algorithm) from MP2 is directly used here, with parameters tuned. For the file transfer, we use RPC, which is similar as MP1 since it is more reliable with lower data loss rate.

1. **Design:**

   Our system has a Master (or Introducer), which keeps track of the file and replica storage information in a table. The Master is also in charge of handling join of new member and reorganize the file storage plans. All other nodes are involved in failure detection.

1.1 Failure detection

Each member will send Ping message to a certain number of its successor nodes and then determine whether the node is failed upon recipient of the responses. More details of this part are already discussed in our report for MP2.

1.2 File/Replica storage

In each VM, we will have a folder for SDFS files and a folder for local files. The local files can then be put into system while the SDFS files can be get into the local folders. The SDFS files are named by its code (can be specified by users), the version, and the time it was created.

In our system, we set up RPC connection at each node to (1) receive files from other VMs, and (2) send file replicas to other VMs. Here we use similar data transfer strategy as what we have in MP1. Also, it should be noted that our file replicas are not stored at Master node, as the Master node is already in charge of failure detection and information transfer, high latency/delay for Master node should be avoided.

Once we need to fetch a file from any VM, a request with the name/version information encoded will be sent to the master, master will then respond with the location information of the VMs with replica, which allows us to setup the connection with those VMs. When we try to put a file from our local directory to SDFS directory, we also inform Master node, which will determine the replica storage plan for the new file.

1.3 Consistency level

In our model, we store at least 4 replicas for each file. The functioning 4 successors for the node that stores the local file will be used as the replica destinations, in which we may be able to distribute the load more evenly. Then once we want to get a SDFS file, we will wait for 3 (= 4/2 + 1) replicas from different VMs and then read the newest file, when we want to put a SDFS file, we will wait for 3 replicas get the newest version, in which we could ensure the quorum consistency.

1.4 Version control

Based on the design, the system will keep the latest 5 versions. Once there are already 5 versions stored for one SDFS file, the new coming version will be stored as the latest version and wipe out the fifth version. Typical get command will return the latest version of the assigned SDFS file.

1.5 Utilization of MP1

We use MP1 result such that we will be able to grep the information for all VMs at one VM, which is much more convenient for debug and status check.,

1.6 Failure and rejoin

Once a node is disrupted, we will then re-replicate all the files it stored to other VMs (based on the functioning successor rule) and modify our file storage table (in our master) accordingly. When any node rejoins the system, we will delete all the old files that are stored in SDFS folders.

2. Test:

2.1 Re-replica time and bandwidth upon failure

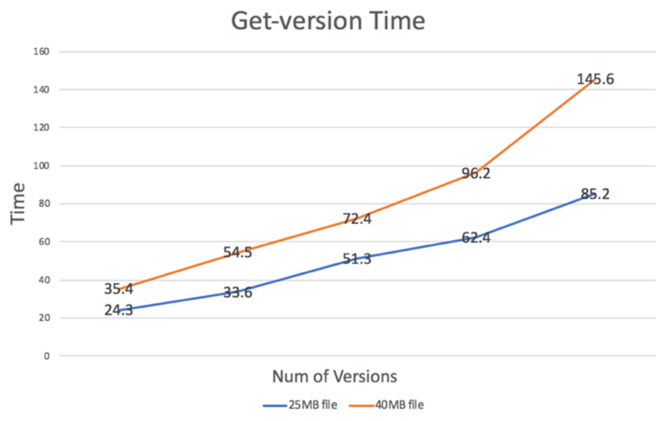
For a 40MB file, the re-replication time is 15~25s, including the detection time, the reorganization time and the file transfer time. The maximum bandwidth is during the file transfer phase, which is 27.2Mb.

2.2 **Insertion/Read/Update time under no failure** (AIT = Average Insert Time; ART = Average Read Time; AUT = Average Update Time; SI = Std for Insert; SR = Std for Read; SU = Std for Update)

File size	AIT (sec)	ART(sec)	AUT(sec)	SI	SR	SU
25MB	8.2	24.3	10.3	0.62	2.06	0.88
500MB	188	594	221	12.4	43.4	17.4

2.3 **Get-version time**

Below is the time for get-version command for different version numbers.



2.4 Time to store the entire Wikipedia English raw text:

After 5 trails, the average time is around 10 minutes.

2.5 Discussion

For the re-replica, the most time-consuming parts are the failure detection (with the time limit for marking a node as failed is 6 sec) and the file transfer phase. We can optimize the failure detection part to reduce the time spent within the first phase, however, this needs to be system/network dependent.

For insert, read and update, the most time-consuming part is the read part which is constrained by the quorum theory. Stand deviation for each of them is acceptable since the max difference is around 20% of the average value. Three replicas are needed to be fetched before we can return the newest version.

For get-version, the more version we need, the more time is required. The trend is almost linear. Besides, we also have constant detection time and instruction transfer time, along with some randomness introduced due to network latency in the total time.

For the storage of Wikipedia English raw text, it's really time-consuming. The design can be improved to avoid any unnecessary I/O to reduce the time.