CS425/ECE428 MP3 Report

Yen-Chieh Sung (ycsung2), Chih-Shin Wang (cswang6)

Design

- Replica
  We set replica number to 4 so that a file will not lost with at most 3 simultaneous failures of nodes. For file **F** corresponding to hash key **K**, we get the first node **i** met by putting all nodes on a ring with the order of their IDs and walking clockwise from **K**, and assign it as the master replica node. Master replica node and its 3 successors (index on the ring +1, +2, +3 from itself) will store replica of file **F** and the version number of **F** (let's say **V[F]**) it has. Master replica node will also hold another sequence # (**S[F].Num**) and the last updated timestamp (**S[F].Timestamp**) to generate new version numbers for update events related to **F**. Note that **S[F].Num** is always >= **V[F]**.
- Versioning for File Update & Delete
  When a client ask to put (insert/update) file **F**, the coordinator server **C** (which can be any node in the member list) will ask sequencer from master replica node for a new version number (**S[F].Num**). If there is no write-write conflict (i.e. <= 1 minute update, which will be explained later), the master replica node will increment **S[F].Num** by one, update **S[F].Timestamp** and return the new value of **S[F].Num**. Then **C** will send update message, including **F** and **S[F].Num**, to all replica nodes of **F**. For a replica node, if its own version number **V[F]** >= **S[F].Num**, then there is no need to update. Otherwise, update the file and the version number **V[F]**. *With this version mechanism, concurrent writes to the same file won't lead to inconsistency among replica nodes*.
  Delete is similar to update. In addition to **F** and **S[F]**, it sends a **delete flag**, preventing inconsistency caused by simultaneously delete and put. The record of **S[F]** and **V[F]** is still kept, but the exact file is deleted, and all replicas will mark **F** deleted. *In other words, we treat delete event as a "new version," so when deleting, we still follow the workflow of put, and identifying the delete event by the* ***delete flag***.
  When the master replica node of **F** changes, the new master replica node will set **S[F].Num = V[F] + 100**, assuming that there will not be more than 100 update events when the replica node set is changing/handling, so that **S[F].Num** from the new master replica node will be large enough for future puts to override previous puts.
  Note that we set buffer size = 1 MB, so client will do partial updates.
- Write-write Conflict
  When requesting a new version number **S[F].Num** from the master replica node, it will check **S[F].Timestamp**. If within 60 seconds, it will reject the update of **S[F]**. Thus, the client will know and prompt the message to the user. We add another "**force update flag**" to do force update if the client confirms the update.
- Failure/Leave/Join Event
  If node **i** leaves/crashes, all nodes will go through all files they're holding. If finding a file's replica set has been changed due to **i**'s absence, compute the new replica node and send the file to it.
  If node **i** joins, all nodes will go through all files they're holding. If finding a file's replica set has been changed due to **i**'s join (i.e. node **i** becomes the replica of the file), send the file to node **i**. Also, some files may be deleted from a node since it may no longer be these files' replica node.
- Read/Write Quorum

Let's consider the following race condition with write quorum size = simultaneous node fail number = 3:

1. A file write to node 1,2,3,4 and get success message from node 2,3,4.
2. Node 2,3,4 fail, and before node 1 gets the update event from step 1's write, it copies old version file to node 5,6,7.
3. Quorum read may get wrong result if node 5,6,7 responds faster than node 1

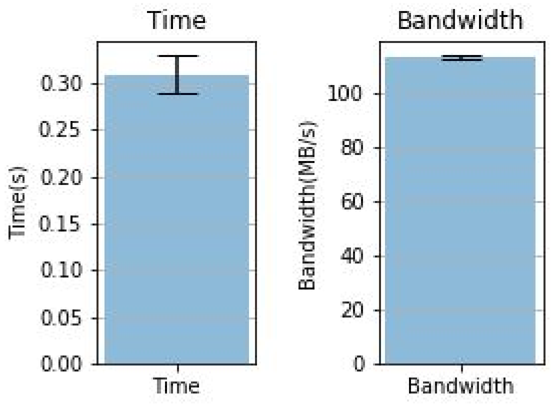
We conclude that write quorum size should be larger than # of simultaneous node failure. To deal with the case above, we set WRITE_QUORUM to 4. READ_QUORUM can be set to 1 consequently so that sum of read/write quorum equals replica_number + 1.

MP1's helpfulness

During testing, we are able to easily fetch logs from every machine by regex/pattern.

Measurement

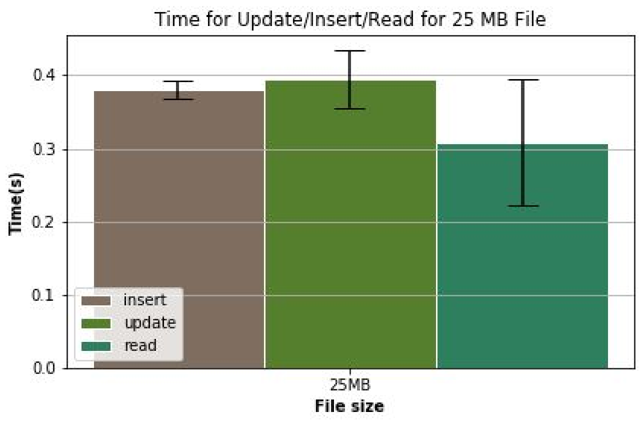
1. Re-replication time and bandwidth upon a failure for a 40 MB file:

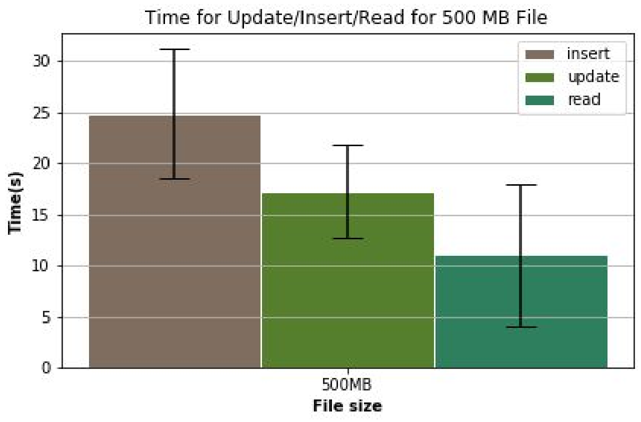


	Avg.	Std.
Time:	0.31(s)	0.0195(s)
Bandwidth:	113.56(Bps)	0.5(Bps)

There should be 3*40 MB data transfer in 0.31 seconds and the maximum bandwidth should be 387MB/s. However, the bandwidth tool seems to provide average of 1 second so the bandwidth here is approximate 3*40 MB/s.

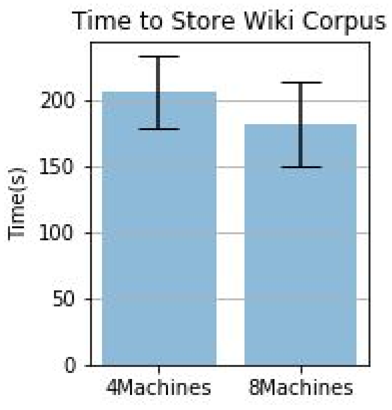
2. Times to insert, read, and update, file of size 25 MB, 500 MB under no failure:





With write quorum size = 4, insert and update are basically the same and thus should have same speed. Since we set read quorum size to 1, read should be faster than the other two. The result meets our expectation, only that for 500MB, update is faster than insert. We think it is just the fluctuation of network speed.

3. Storing English Wikipedia corpus with 4 & 8 machines:
The figure shows that storing files on 8 machines is much faster than storing on 4 machines. We guess this observation might be related to network load balancing. The cause may also related to the network state when testing since 4 machines has a record of runtime faster than some of 8 machines, and max runtime of 8 machines is almost same as the max runtime of 4 machines.


Time to Store Wiki Corpus
Time(s)
4Machines
8Machines