Milestone 3: Distributed & Replicated Storage Service

Team Info

- Xuan Chen(1002344732)
- Tian Tian(1002556138)

Replication Mechanism

Coordinator's Responsibility

The replications are managed and issued by each coordinator. In our design, the coordinators are responsible for synchronizing the "PUT" requests with its replicas through the network using "PUT_REPLICATE". For the shared range between these three storage servers (i.e. coordinator's hash range between its predecessors and itself), only the coordinator will handle the "PUT" request while all three nodes can handle the "GET" request.

Use of Zookeeper to manage the grouping of 1 coordinator & 2 replicas

The ECS service will perform actions including adding or removing a node from the hash ring and save the updated hash ring (i.e. *"/metadata"*) in a znode that is accessible by all storage servers. When initializing the storage servers, each node will set a watcher on the znode that tracks the latest hash ring. Therefore, whenever an update happens on this znode, each server will "process" this change to re-identify its replicas that will be synchronized with its own PUT requests.

System Reconciliation under scaling

In order to maintain the replication invariant (i.e every data item is replicated on the two and only two successor storage servers of the coordinator), the following data transfers are performed when adding or removing a server node.

Adding Node

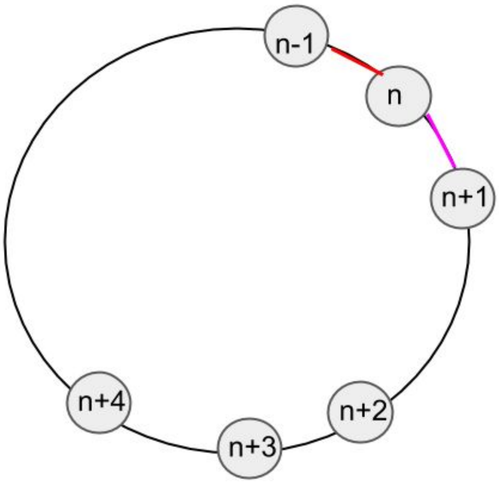


Figure 1: Before the addition of a node

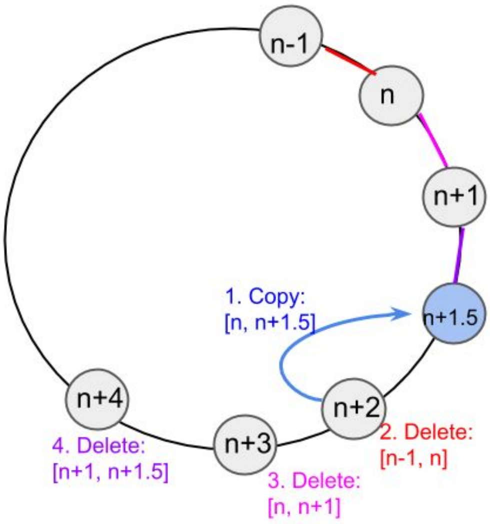


Figure 2: When adding node (n+1.5)

Responsibility Change of hashing range due to addition of node (n+1.5)

Node	n+1.5	n+2	n+3	n+4
Before addition	N/A	[n-1, n+2]	[n, n+3]	[n+1, n+4]
After Addition	[n-1, n+1.5]	[n, n+2]	[n+1, n+3]	[n+1.5, n+4]

Table 1: Responsibility change of hashing range due to node addition

Based on the recalculation of changes of each node's responsible range, the ECS notifies the affected four nodes to make one copy and three deletions using zookeeper's znodes. The copy action will happen first and then the three deletions with no order preference. Only the "copy" action will establish a socket connection between the two nodes, (n+1.5) and (n+2). The deleters will know the range of deletion from each's znode and delete the KV pairs locally without interruptions. This manner minimizes the use of network throughput for data transfer.

Removing Node

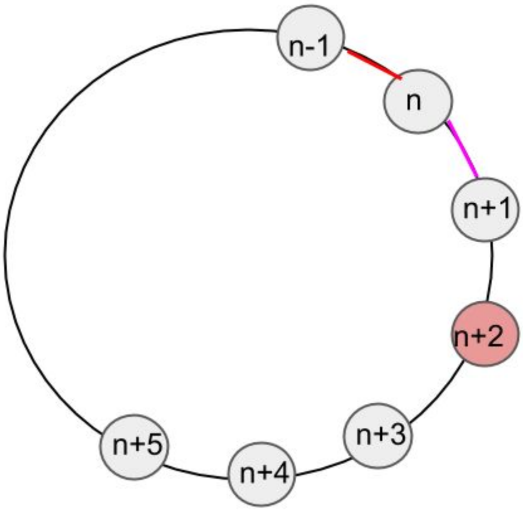


Figure 3: Before the removal of the node (n+2)

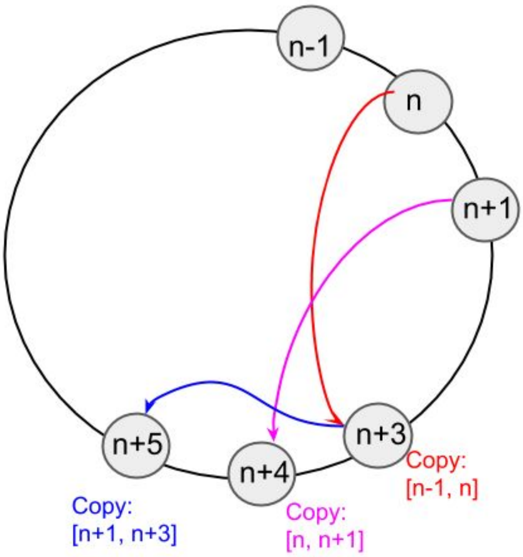


Figure 4: After removing the node

Responsibility Change of hashing range due to the removal of the node (n+2)

Node	n+2	n+3	n+4	n+5
Before addition	[n-1, n+2]	[n, n+3]	[n+1, n+4]	[n+2, n+5]
After Addition	None	[n-1, n+3]	[n+1, n+4]	[n+1, n+5]

Table 2: Responsibility change of hashing range due to node removal

When removing a storage server from the consistent hash ring, three copy commands are individually sent through zookeeper's znodes (i.e. */server/${portNo}/op*) to the three affected senders(i.e. Node (n), (n+1), (n+3)) along with the information of the receiver identifier and the copied range. The data transfer will happen through three socket connections between these three new pairs of coordinator and the last replica. The storage of the deleted node (n+2) will be cleared upon receiving the "SHUT_DOWN" operation from the zookeeper (i.e. the same as M2's implementation) so additional commands will be issued.

Failure detection and recovery

Failure Detection

When initializing each storage server for node addition, each node will create a znode for itself of Create Mode, EPHEMERAL (i.e. Under */failure_detection*). After initialization is done, the ECS will be notified via zookeeper (i.e. deletion of operation message's znode) and then set a watcher on this newly-created znode. Therefore, once this storage server crashes, this znode will be deleted automatically due to the end of this zookeeper session. The ECS will determine whether it is an "actual" failure or a normal node deletion by looking at whether the node is still on the hash ring. If so, it is an actual crash.

Failure Recovery

The ECS will remove the crashed servers from the hash ring and add it back to the list of available servers. The crashed server will be recorded under zookeeper by creating a PERSISTED znode under the directory *"/crashed"* so that its storage will be cleared when re-initializing (for the sake of consistency). Then it will perform the data transfers which are the same as when removing a node. Afterwards, it will add another node into service to recover its effect on the load distribution.

Client-side Storage failure handling

A client detects a server failure by catching an IOException due to request socket error or an Exception due to infinite message receiving loop. The client first disconnects from the failed server. The client then looks at its latest version of the metadata information of the hash ring (if received previously together with a "SERVER_NOT_RESPONSIBLE" error), and connects to the deemed responsible server to retry the client request. If no server information is available, the client prompts the user about this connection loss.

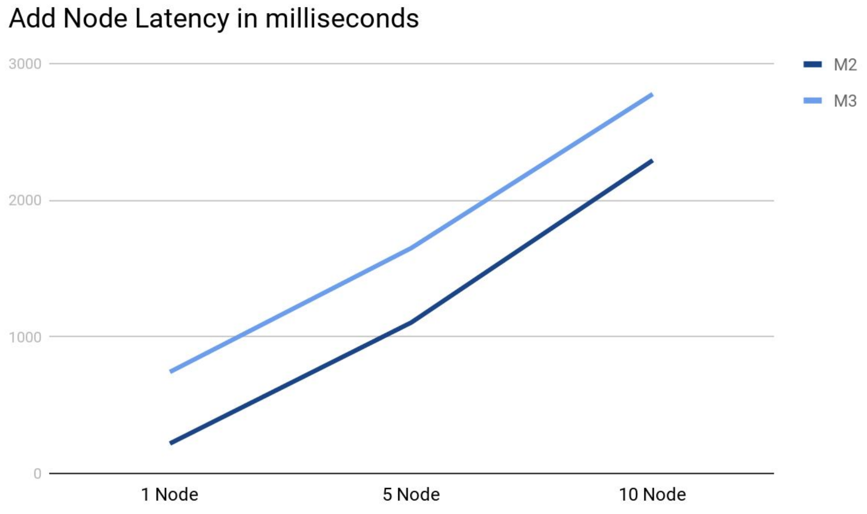
Performance Evaluation

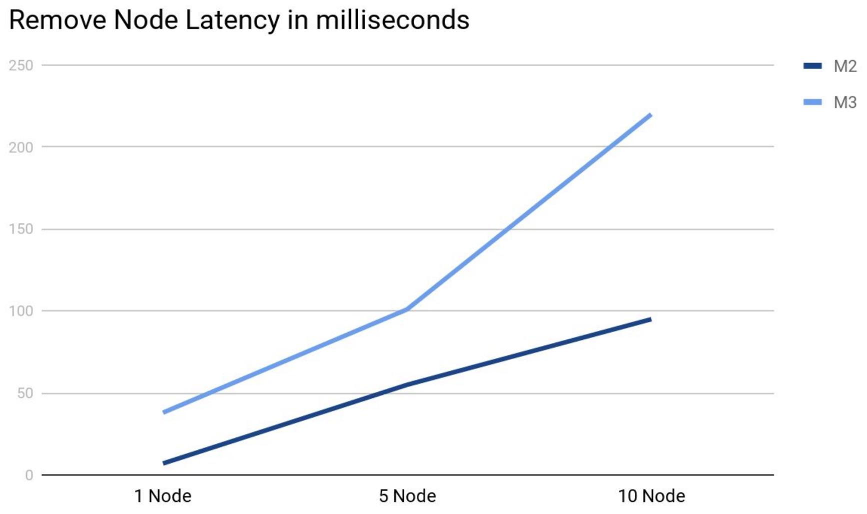
M3 performance is evaluated using the following metrics:

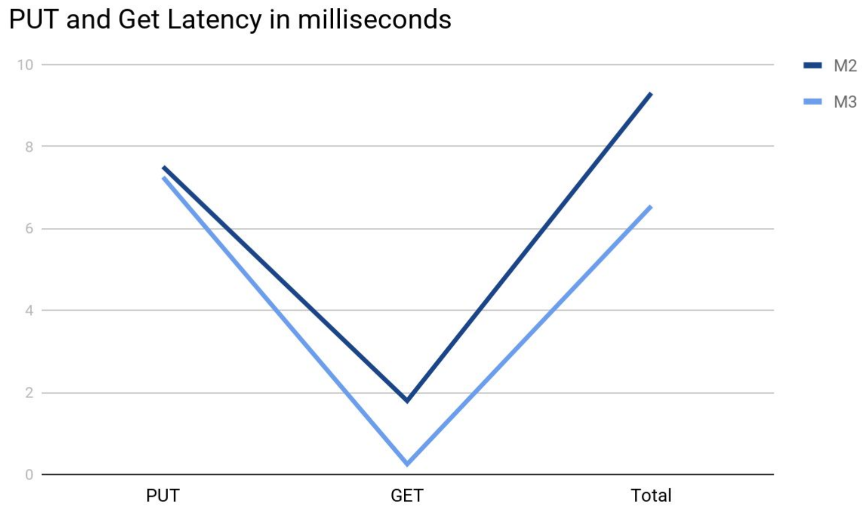
- The latency of add/remove of 1, 5, and 10 nodes/servers
- The latency of Put/Get using the Enron Email Data set

As seen in the appendix section below, the overall performance of M3 is worse compared to M2 for adding and removing nodes. This is expected as the replication mechanism implemented in this milestone contributes greatly to the additional latency. Both adding and removing servers trigger the replication process, thus, performance is traded off against availability. Although the latency increased due to replication, data throughput is increased as a result of the high availability from eventual consistency. Aside from these main differences, the two milestones share a similar trend in its performances. This is also expected as the main framework (ECS, ZooKeeper, Hash Ring) of our design is unchanged.

Appendix A: M2 & M3 Performance Comparison







Appendix B: Milestone 3 Unit Tests

Test Name	**test01Replication**
Status	Pass
Description	Test Key-Value pair replication across replicas

Test Name	**test02ReplicationAfterAddNode**
Status	Pass
Description	Test Key-Value pair replication when adding new nodes

Test Name	**test03ReplicationAfterRemoveNode**
Status	Pass
Description	Test if predecessors become new replicas if the original replicas are removed

Test Name	**test04NodeFailureDetection**
Status	Pass
Description	Test if server crashes are identified by the ECS

Test Name	**test05NodeFailureRecovery**
Status	Pass
Description	Test crashed server can be recovered after a failure detection

Test Name	**test06ReplicaGet**
Status	Pass
Description	Test if replicas can successfully process GET command

Test Name	**test07ReplicaPut**
Status	Pass
Description	Test if replicas can correctly handle PUT command

Test Name	**test08HashRingAddingNode**
Status	Pass
Description	Test if the hash ring can identify the correct replicas when nodes are added

Test Name	**test09HashRingRemovingNode**
Status	Pass
Description	Test if the hash ring can identify the correct replicas when nodes are removed

Test Name	**test10Consistency**
Status	Pass
Description	Test if data changes such as PUT/PUT_UPDATE/DELETE that happened during a server crash are reflected in the recovered server side.

Appendix C: Previous Unit Tests

M2

Test Name	**test_createECS**
Status	Pass
Description	Test the creation of ECS

Test Name	**test_addNode**
Status	Pass
Description	Test adding nods to ECS

Test Name	**test_startNode**
Status	Failed
Description	Test if the previously added nodes can be started.

Test Name	**test_removeNode**
Status	Pass
Description	Test if all the nodes can be removed from ECS

Test Name	**test_removeNonExistNode**
Status	Pass
Description	Test if removing a non-existed node is handled

Test Name	**test_stop**
Status	Pass
Description	Testing if ECS can be stoped

Test Name	**test_shutdown**
Status	Pass
Description	Test if ECS can be shutdown

Test Name	**test_connection**
Status	Pass
Description	Test if clients can connect to ECS started servers

Test Name	**test_putGetData**
Status	Pass
Description	Test if the client can put and get data from ECS stared servers

Test Name	**test_addNodes**
Status	Pass
Description	Test if a node can be added to hash ring

Test Name	**test_getNode**
Status	Pass
Description	Test if the hash ring can return a node by name and hash

Test Name	**test_removeNode**
Status	Pass
Description	Test if a node on the hash ring can be deleted

M1

Test Name	**test_Set_Value**
Status	Pass
Description	Test set KEY VALUE pair function

Test Name	**test_Get_Value**
Status	Pass
Description	Test get VALUE with a given KEY function

Test Name	**test_Update_Value**
Status	Pass
Description	Test update VALUE with a given KEY function

Test Name	**test_Get_non_exist**
Status	Pass
Description	Test if the server responds correctly with a getting a non-exist KEY

Test Name	**test_Put_LongStr**
Status	Pass
Description	Test if the server responds correctly with putting a very long string

Test Name	**test_Delete_Pair**
Status	Pass
Description	Test delete KEY VALUE pair with a given KEY

Test Name	**test_Delete_non_exist**
Status	Pass
Description	Test if the server responds correctly when deleting a non-exist KEY

Test Name	**test_FIFO_func**
Status	Pass
Description	Test if the FIFO cache strategy is working

Test Name	**test_LRU_func**
Status	Pass
Description	Test if the LFU cache strategy is working

Test Name	**test_cache_perf**
Status	Pass
Description	Test the throughput and latency of the server with various cache size and number of PUT/GET requests

Test Name	**test_Connection**
Status	Pass
Description	Test if the client can make a connection with the server