University of Toronto
Faculty of Applied Science and Engineering

ECE419
Milestone 3 Report

Team ID	B7
Date	March 25th, 2018
Project Title	ECE419 - Milestone 3
Prepared By (Names and Student #s of Team Members	Aya Elsayed [1000733469] Angel Serah [1000491773] Harshita Huria [1000980398]

In this document, we provide implementation and design decision details for the third milestone of ECE419 project. For milestone 3, we are required to maintain replicas for each of the servers so that if one of the servers crashes, we still have 2 copies of the data and can still respond to client get/put requests about this data. We are also required to actively detect crashes and start a new server to replace a crashed one.

Our Implementation

KVStore:
We made a few changes to the client side such that if a client is connected to a server not responsible for a key, the client can then randomly connect to and read from the either the coordinator (who is responsible for the key) or their replicas. The way we implemented this functionality is:

- If the server the client is connected to responsible for the key, it will get it for the client.
- If the server the client is connected to is not responsible, the client requests metadata from the server so it can figure out who the responsible server is. The server send back the metadata as well as a list of replicas associated with each server. The client then *randomly* chooses from the coordinator (who is responsible for the key) and their replicas to request the key from.

KVServer:
The main changes we made to KVServer include adding two replicas (if there are enough nodes on the network) for each server. These replicas are the first and second successors of the coordinator server. Each server's replica is updated immediately when a change takes place in the coordinator. We added in separate files to keep track of the data a server is responsible for as a coordinator, a primary replica and a secondary replica. As new nodes are added or removed, and the replicas changed, the files are updated with the new values from the replicas' new coordinators.

ECS
There were two major changes for the ECS.

- As ECS adds new nodes to the network, it will update the servers on who their new replicas are. This is done by adding the node to the network and then for each node, going around the circular network, finding the 1st and 2nd successor and updating that node with this information.
- The ECS is also able to detect crashes and failures and recover from them. This is done through the zookeeper. KVServer spawns a new thread on its initialization which updates its znode with the current time every 5 seconds. ECS checks through the znodes of each server and compares the value there to the current time. If it is not within 8 seconds of each other, then that KVServer is considered crashed. If it considered a crashed server, ECS removes that node and adds a new one to the network if there are any more available servers.

Unit Tests & Performance Evaluation
Please refer to Appendix A for a table of tests added.

Appendix A

Unit Tests:

Test #	Functionality Tested
1	Using ECS, init 1 server, kill the server process, make sure ECS realized the server has crashed.
2	Init 3 servers, add KV pairs such that each server is responsible for at least 1 KV pair, ensure that the replica files of each contain the expected data
3	Init 2 servers, add KV pairs such that each server is responsible for at least 1 KV pair, ensure that the replica files of each contain the expected data (corner case of test 2)
4	Init 3 servers, add KV pairs such that after all key redistribution, only 2 out of the 3 servers coordinators FILES contain data. Ensure that its primary and secondary replicas' files are now empty.
5	Using ECS, init 2 servers, add a KV pair to one of them. Kill that server's process. Init a client, connect to the second server and get the KV pair that was put earlier. Connect to the crashed server's replacement and get the KV pair that was put earlier. Both should have it, either as a coordinator or a replica.
6	Init 3 servers, add KV pairs, ensure that the replica files of each contain the expected data. Remove 1 server and check that all files (coordinator and replicas) still contain the expected data.
7	Init 2 servers. Connect a client to one and put a KV pair which hashes to it. Disconnect and connect to the other server (which is not responsible for that KVpair). When key is read, client should not get reconnected to the responsible server but receive a GET_SUCCESS .
8	Make sure we only have 2 available servers. Init both servers. ECS should handle crash of all nodes on the network and replace it with exactly 2 servers. Check if there are exactly 2 in the network.
9	Init 2 servers. Connect a client to one and put in a KV pair which doesn't hash to it. Should get SERVER_NOT_RESPONSIBLE and get reconnected only to the coordinator (responsible server). Gets a PUT_SUCCESS.
10	Init 2 servers. Connect a client to one and put in a key. Disconnect client. Kill them all so all the servers in the network have crashed. ECS should replace them with 2 new servers. Reconnect a client and get the key. Gets a GET_SUCCESS.
11	**Init 4 servers. Make sure that all replicas have the expected data. Each server won't be a replica of at least a server on the network.**

Performance Evaluation

Using a few selected KV pairs, performance testing was done for the parameters outlined by the table below which reports the latency of our system.

Tested with trying to put and get 300 emails

Num of clients/ servers	1 server	5 servers	20 servers	Cache config
1 client	7.73s	9.98s	11.23s	FIFO, 5
	7.77s	10.20s	10.59s	LRU, 5
	8.99s	9.65s	11.60s	LFU, 5
5 clients	8.34s	10.46s	12.38s	FIFO, 5
	8.13s	10.47s	12.40s	LRU, 5
	7.83s	10.18s	12.98s	LFU, 5
20 clients	8.36s	11.15s	13.43s	FIFO, 5
	7.80s	11.33s	13.11s	LRU, 5
	7.77s	11.15s	13.90s	LFU, 5