ECE419 Distributed Systems

==========

Team 69

Milestone 2 Design Document

==========

Team

Amar Arefeen

Arjun Mittal

Ravi Singh

Submission Date

February 26th, 2021

Design

The following document will go over the team's implementation of the external configuration service (ECS), as well as any changes to KVServer, or any of the surrounding modules to support ECS. Design decisions, performance, and testing will be reviewed as well.

New Additions

The communication flow is depicted in Figure 1. Communication starts at the ECS CLI which is responsible for taking user input and calling the appropriate ECS function. All management of ZooKeeper is handled by ZooKeeperService, a wrapper class around ZooKeeper. The ECS structure is implemented as an ECS Hash Ring of ECS Nodes (ZooKeeper Nodes). ECS Server Communication then handles any requests to a server's ECS Nodes, and metadata updates. The corresponding response is then propagated back to the ECS CLI.

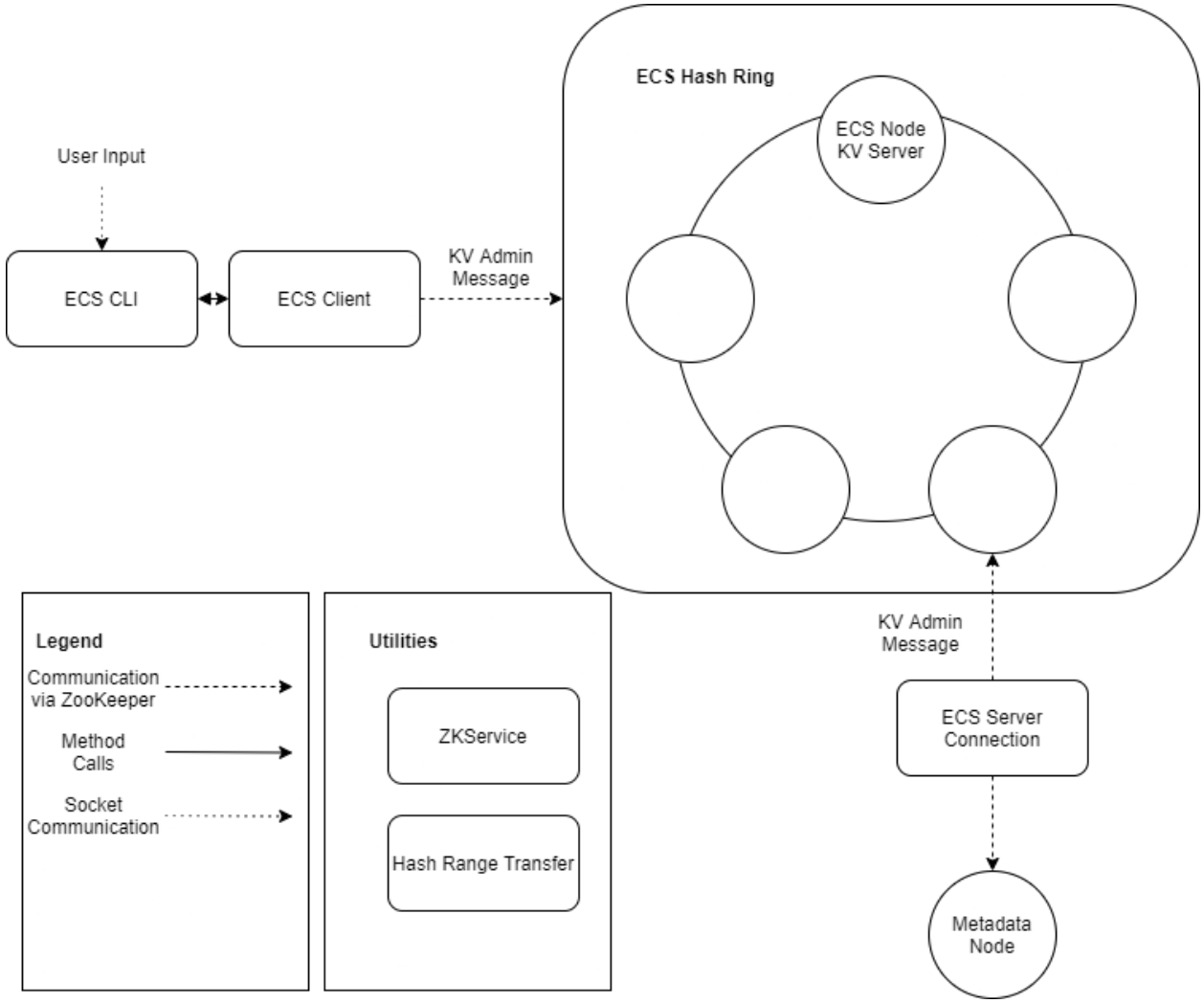


Figure 1: ECS Architecture Overview.

ECS CLI

This is a command line interface that implements the commands for ECS. Messages are encapsulated in a KVAdminMessage type similar to how messages between KVServer and KVClient are encapsulated as KVMessages. The implementation of the CLI does not differ greatly from the KVClient CLI in M1.

ZooKeeper Service

This class is the backbone of the communication infrastructure. ZooKeeper Service is responsible for implementing functionalities related to node creation, deletion, modification, as well as setting watchers on these nodes. To manage watchers, and any associated asynchronous callbacks the team maintains a thread pool here. There are two types of watch use cases the team considered. The first is a persistent watch which would essentially be for events such as metadata updates. As watches are one time events, the triggering event is proxied to the appropriate handler and the watch is reset. Alternatively, there is the one time watch used by the message sender to wait on replies.

ECS Node

The ECS Node class represents one node in our ECS Hash ring i.e. a singular KV Server object. An extension of this is the ZKECSNode which also maps to a zNode in our hash ring, so we can use it for communication. A persistent watch is set on this object to allow it to listen for requests coming from ECS. This object is also responsible for maintaining key range, hash range, predecessor information, which are used during the transfer process and determining whether the node is responsible for a request.

ECS Hash Ring

This overarching structure is the entire cluster of ECS Nodes being managed by ECS. The team chose to implement this as a sorted map. The sorting is based on the key ranges the nodes are responsible for.

ECS Server Communication

ECS Server Communication is responsible for actually handling the responses to the requests sent from ECS CLI. This module is semi-stateless as ECS CLI takes care of managing request flow. Care is taken to prevent a feed-back loop of occurring where the responsible node picks up both the request and the reply to the request, resulting in incorrect behaviour. Transfer is done via the transfer utility class. Two callables (threads) are used to ensure simultaneous transmission of the TRANSFER_BEGIN message. These threads are also responsible for receiving the TRANSFER_COMPLETE from ECS Server Communication.

Modifications

There were a few changes made to KVServer in this milestone. State was added to KVServer to determine whether it is ready to respond to requests, locked, stopped, etc. Additionally, accessor methods to retrieve the server metadata were added. All additional interactions with ECS were handled by the ECS Server Connection class which was instantiated as a new member of KVServer. A minor bug fix to server shutdown was made in which client sockets were not being closed correctly.

In KVStore modifications were made to handle retry attempts and server failure. Specifically, there are two scenarios that the team has accounted for:

1) Server outage
2) Node removal

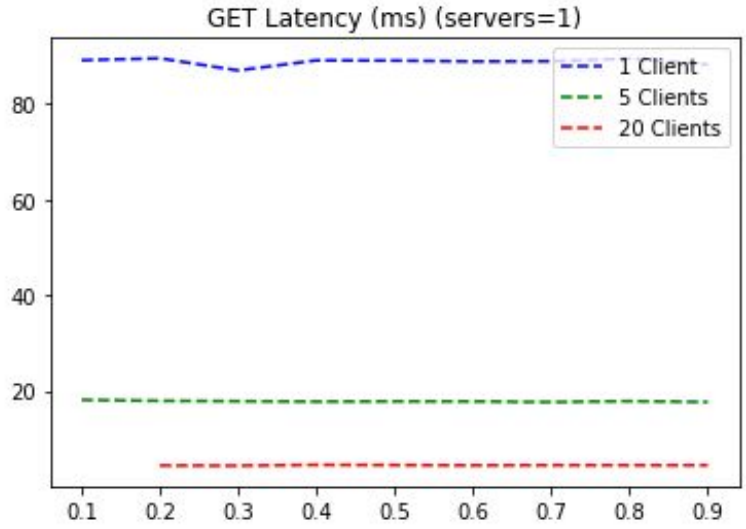
However, the end result is identical. On server outage or node removal, KVStore continues to retry the same request three times. Upon three successive missed ACKs, KVServer returns an error message. At this point, KVStore talks to an arbitrary server which consequently responds with a metadata update. Assuming successful transfer, the metadata should now show a new server responsible for the key(s) of interest.

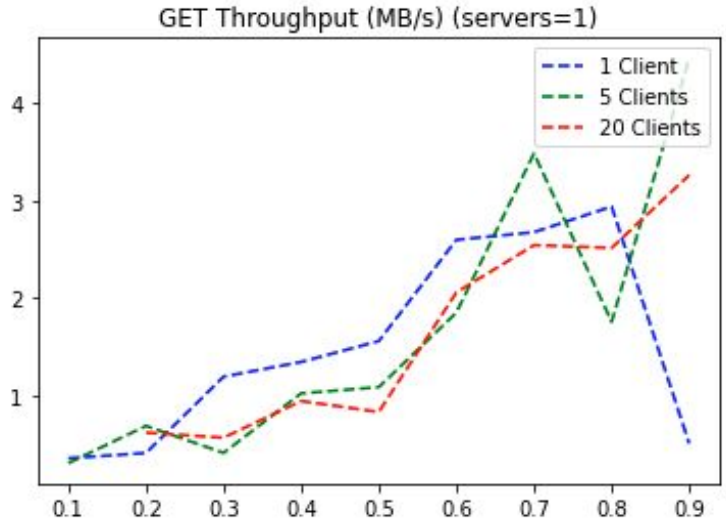
Performance Evaluation

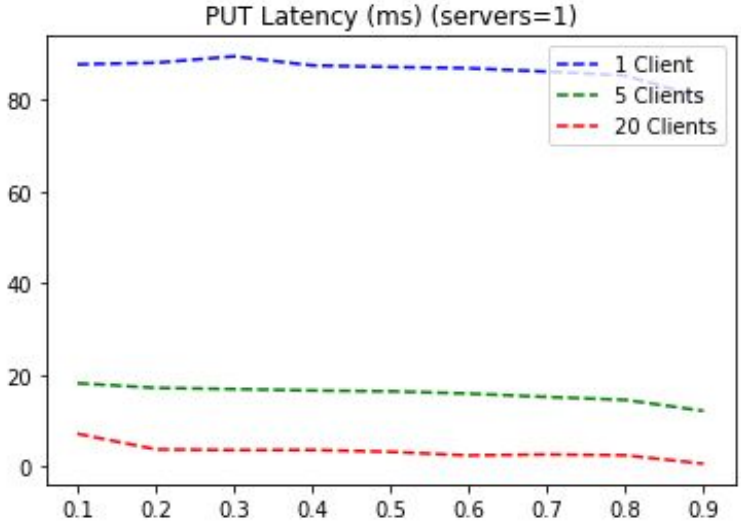
For performance evaluation, the team tested the latency and throughput for both get and put requests. The plotted variables on the next page were the GET/Request ratio (x axis), number of concurrent clients (different lines), and number of servers in the ECS ring (different graphs).

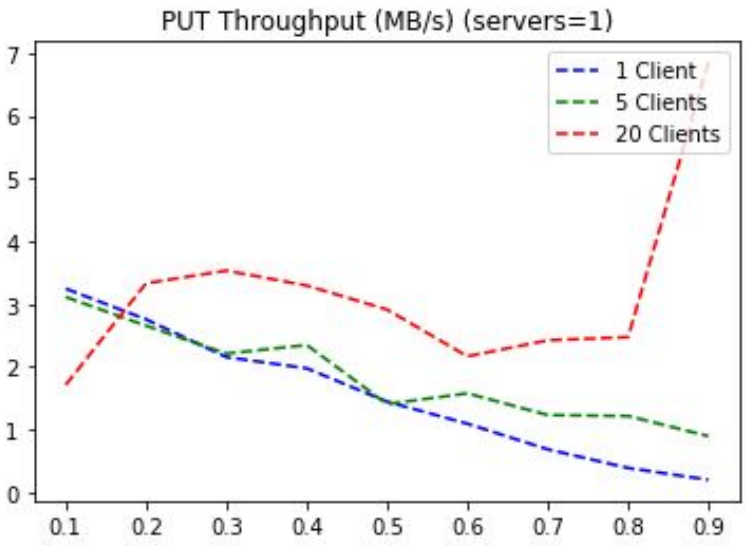
The tests were run on a UG machine. Unfortunately, these testing conditions were not ideal and it is hard to make meaningful observations from this data, but we can at least see that performance across different GET/Request ratios is consistent, as is the performance across the varying number of clients.

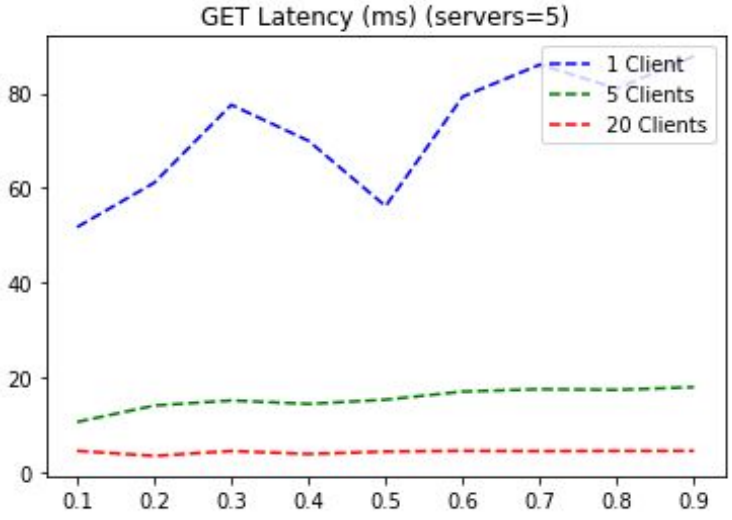
Presented on the following pages are some simple graphs for our performance.

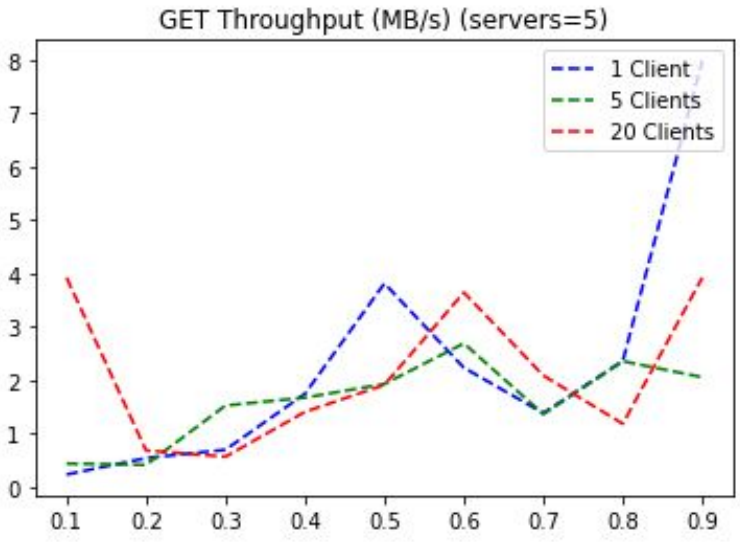

GET Latency (ms) (servers=1)
1 Client
5 Clients
20 Clients
80
60
40
20
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

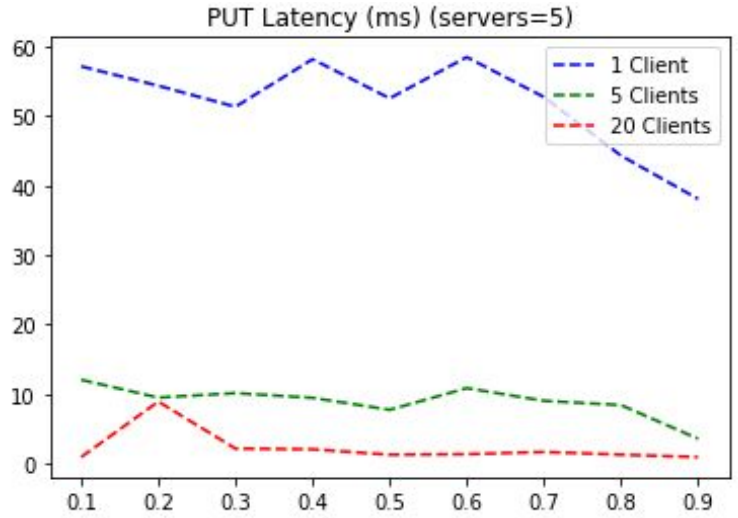

GET Throughput (MB/s) (servers=1)
1 Client
5 Clients
20 Clients
4
3
2
1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

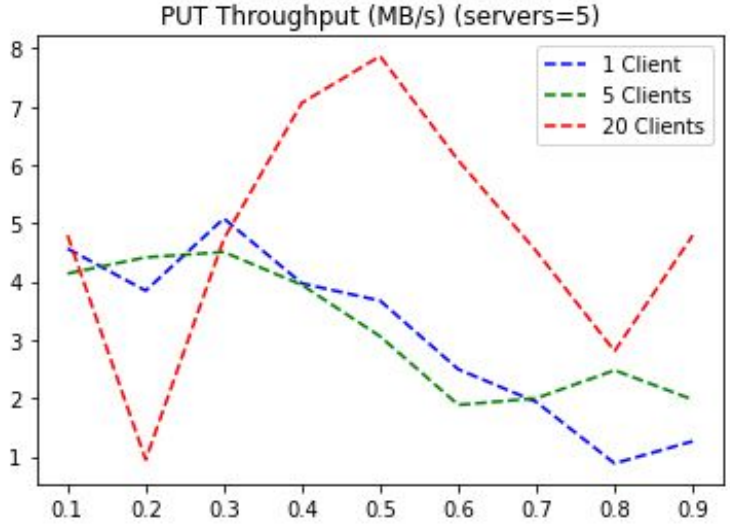

PUT Latency (ms) (servers=1)
1 Client
5 Clients
20 Clients
80
60
40
20
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


PUT Throughput (MB/s) (servers=1)
1 Client
5 Clients
20 Clients
0
1
2
3
4
5
6
7
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


GET Latency (ms) (servers=5)
1 Client
5 Clients
20 Clients
0
20
40
60
80
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


GET Throughput (MB/s) (servers=5)
1 Client
5 Clients
20 Clients
0
1
2
3
4
5
6
7
8
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


PUT Latency (ms) (servers=5)
1 Client
5 Clients
20 Clients
0
10
20
30
40
50
60
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


PUT Throughput (MB/s) (servers=5)
1 Client
5 Clients
20 Clients
1
2
3
4
5
6
7
8
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

Appendices

The following section outlines all tests, their relevant status and description which ensures the correct functionality of our application.

Appendix A: Existing Unit Tests

Category	Connection Test
Name & Status	testConnectionSuccess - Pass ✔️
Description	Verifies a connection between client and server can be established.

Category	Connection Test
Name & Status	testUnknownHost - Pass ✔️
Description	Verifies an UnknownhostException is thrown when connecting with an invalid address.

Category	Connection Test
Name & Status	testIllegalPort - Pass ✔️
Description	Verifies the specified connection request contains a port in the appropriate range.

Category	Interaction Test
Name & Status	testPut - Pass ✔️
Description	Verifies a put request can be executed without an exception.

Category	Interaction Test
Name & Status	testPutDisconnected - Pass ✔️
Description	Verifies a put request throws an exception when a client is not connected.

Category	Interaction Test
Name & Status	testUpdate - Pass ✔️
Description	Verifies a put request with an existing key in the database report's status of PUT_UPDATE.

Category	Interaction Test
Name & Status	testDelete - Pass ✔️
Description	Verifies a put request with an existing key and a null value, deletes the existing key from the database and report's status of DELETE_SUCCESS .

Category	Interaction Test
Name & Status	testGet - Pass ✔️
Description	Verifies a get request returns the appropriate associated value from a previous put request.

Category	Interaction Test
Name & Status	testGetUnsetValue - Pass ✔️
Description	Verifies a get request reports a status of GET_ERROR if the associated key does not exist in the database

Appendix B: Additional Unit Tests

Category	Additional Test
Name & Status	testFifoCache - Pass ✔️ testLruCache - Pass ✔️ testLfuCache - Pass ✔️
Description	Verifies specific caching eviction policy is working as intended.

Category	Additional Test
Name & Status	testOneNodeHashRing - Pass ✔️
Description	Verifies addition and removal of one node to hash ring data structure. Checks to see if the integrity of the hash ring is maintained.

Category	Additional Test
Name & Status	testTwoNodeHashRing - Pass ✔️
Description	Verifies addition of two nodes, removal of one from hash ring data structure. Checks to see if the integrity of the hash ring is maintained.

Category	Additional Test
Name & Status	testMultiNodeHashRing - Pass ✔️
Description	Verifies addition of three nodes (generalized case), removal of two nodes from hash ring data structure. Checks to see if the integrity of the hash ring is maintained.

Category	Additional Test
Name & Status	testHashRingSerialization - Pass ✔️
Description	Verifies the hash ring is serialized correctly. Expects order and format to match.

Category	Additional Test
Name & Status	testSendReceive - Pass ✔️
Description	Verifies sending and receiving of KVAdmin messages to/from ECS.

Category	Additional Test
Name & Status	testAddNodes - Pass ✔️
Description	Tests addition of multiple nodes to ECS (different from hash ring), implicitly tests addNode.

Appendix C: Performance Tests

Category	Performance Tests
Name & Status	N/A
Description	Please refer to the Performance Evaluation section above to find all tests related to how we tested the performance of our system. Alternatively, the code is available in src/testing/PerformanceTests.java