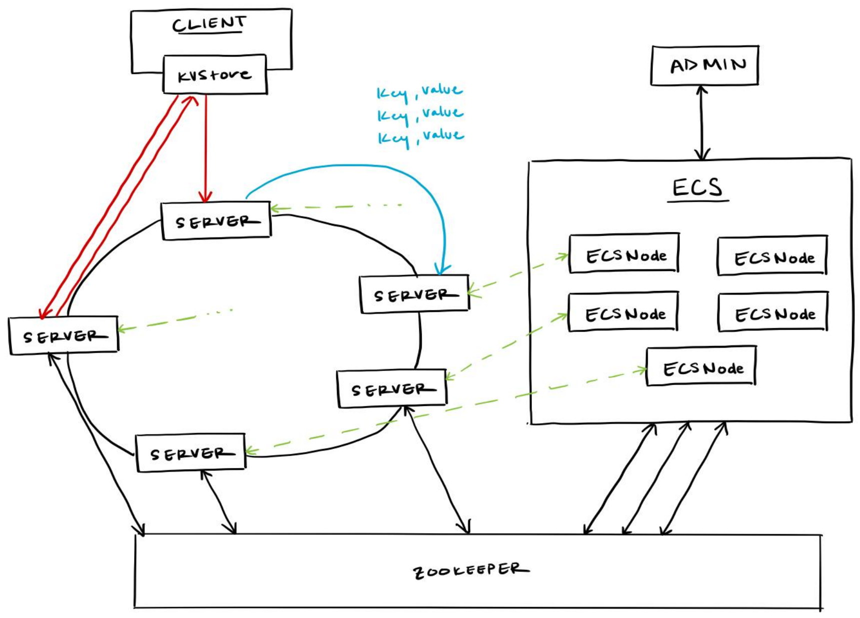
ECE419 Milestone 2 Design Document

Team 28: Akino Watanabe (1004133653), Lisa Li (1003924532), Sandra Petkovic (1004018537). Submission by: *petkov12*

The Overall Design



Design Decisions

ECS & HashRing

The ECS manages the distributed storage service (DSS) by allowing an administrator to add/remove nodes, start and stop the service, and configure ZooKeeper for each node. The ECS stores all possible servers as well as a list of active servers. The ECS passes this information to a HashRing class that handles the DSS logic and messaging for all nodes.

Each server node in the hash ring is represented by an ECSNode. The ECSNode contains information about the server, such as its hash range and port, as well as an ECSConnection. HashRing uses this to communicate with each node by calling `node.sendMessage(msg)`. Each message sent by the HashRing contains a status, key, value, and Metadata field (see **Messages**). The metadata field stores an unordered dictionary of server names and hash keys for the server to determine its position in the hash range. At each update of the hash ring, every node is sent an UPDATE_METADATA message with the new order. The node that must transfer data receives a MOVE_DATA message status with additional information on a "receiver node" to inform the KVServer where it must transfer its data.

ZooKeeper

ZooKeeper monitors the heartbeats of online servers. When a server is added, it creates an ephemeral node on the path "/heartbeat/<server name>". The ECS then waits for a NodeCreated event before it proceeds. Similarly, when a server shuts down, the ECS waits for a NodeDeleted event before continuing with the rest of its protocol.

Messaging

In Milestone 1, the JSONMessage class manually serialized/deserialized JSON with the structure: `{“status”:”PUT”,”key1”:”value1”}`. For Milestone 2, we modified it so Gson handles the serialization/deserialization, and we included an additional *metadata* field:

```
{“status”:”PUT”,”key”:str,”value”:str,“metadata”:null || Metadata.class}
```

where the Metadata class has the following structure, simplified:

```
{“status”:”START”,”order”:hashring type,”receiverNode”:null || ECSNode type}
```

Metadata’s *status* contains actions for the KVServer to perform (e.g. move data to another server, update its metadata); the *order* is the current, unordered state of the hash ring; when a data transfer is required, *receiverNode* is populated with the receiving server’s information.

A KVServer uses the same ServerSocket connection on its own port to listen for messages from the ECS, KVClient, and other KVServers. The server determines the message type by checking if the *metadata* field in JSONMessage is null. If so, it understands that the message is from KVClient (or another server, see **KVServer & Moving Data**), and calls *handleMessage()* to execute the proper commands from Milestone 1. If *metadata* is not empty, the server knows this is an admin message from the ECS and calls *handleMetadataMessage()* to perform the proper command.

In the case where a server determines it’s not responsible for a KVClient’s command, the server will return a SERVER_NOT_RESPONSIBLE JSONMessage with the *metadata* field containing the current hash ring. Likewise happens with a SERVER_WRITE_LOCK message when the server is locked for write.

KVServer & Moving Data

KVServer has been modified from Milestone 1 to accommodate the metadata it receives. Upon receiving new metadata from the ECS, the server determines its end hash from its placement in the ring. When a KVServer receives a MOVE_DATA message with *receiverNode* fields, it checks if its own name is in the current ring. If so, it knows to only transfer certain key-value pairs that no longer fit within its range. Otherwise, if its name is not in the hash ring, meaning it has been removed, it will transfer all its key-value pairs to the receiver node then kill itself.

To send the data, the KVServer opens a new socket to connect to the receiver node and sends the key-value pairs in a PUT_MANY JSONMessage. Once the receiver node receives the message, it appends the new key-value pairs to its own storage.

KVClient & KVStore

When receiving a SERVER_NOT_RESPONSIBLE response and the updated hash ring data, the KVStore will determine the correct server from the ring, connect to that server, and retry the request. This switching process is invisible to the client and appears as if the correct server was the initial contact. The KVStore stores this hash ring state and updates it whenever it receives new metadata.

Performance test

Number of nodes (no caching)	Time duration of adding nodes (ms)	Time duration of removing nodes (ms)
1	1126	2
3	3245	16
5	9332	19
7	9705	58
10	15687	87

Number of servers	Number of clients	Caching strategy	Cache size	Latency (ms)	Throughput (B/ms)
1	2 (8 requests)	None	0	6260000	1.5974E-3
3	2 (8 requests)	None	0	6263625	1.5965E-3
4	2 (8 requests)	None	0	13766125	6.3562E-4
2	1 (8 requests)	None	0	8752250	8.5692E-4
2	2 (8 requests)	None	0	6261750	1.5969E-3
2	4 (8 requests)	None	0	27510125	9.0875E-5
2	1 (8 requests)	FIFO	10	10637125	9.4010E-4
2	1 (8 requests)	LFU	10	10648750	9.3907E-4
2	1 (8 requests)	LRU	10	8750750	8.5706E-4
2	1 (8 requests)	FIFO	3	10638750	9.3996E-4
2	1 (8 requests)	FIFO	10	10640125	9.3983E-4
2	1 (8 requests)	FIFO	20	10641000	9.3976E-4

Attribution

Name	**Contributions to M2**
Akino Watanabe	Test cases, Performance Evaluations, KVStore, KVServer caching, KVClient—KVServer communication, Design document, Debugging
Lisa Li	Test cases, PersistentStore, ZooKeeper, ECSClient, ECS—KVServer communication, KVServer—KVServer communication, Design document, Debugging
Sandra Petkovic	Test cases, HashRing, ECSClient, ECSNode, ECSConnection, Metadata, ECS-KVServer communication, KVServer—KVServer communication, Design document, Debugging

Appendix

Appendix 1: Unit tests

Cache Tests

Test Function	**Description**
FIFOCacheTest	Confirms that the FIFOCache works appropriately.
LRUCacheTest	Confirms that the LRUCache works appropriately.
LFUCacheTest	Confirms that the LFUCache works appropriately.
singleServerRingTest	Confirms that a hash ring can be initialized with one server.
multipleServerRingTest	Confirms that a hash ring can be initialized with multiple servers.
removeServerTest	Confirms that a node can be removed from a hash ring.
removeServersTest	Confirms that multiple nodes can be removed from a hash ring.
testECSStart	Confirms that a client can make a PUT request to a started server.
testECSStop	Confirms that a client cannot make a request to a stopped server, server responds with SERVER_STOPPED.
testServerNotResponsible	Confirms that a server responds with SERVER_NOT_RESPONSIBLE when a client makes a PUT request with an out-of-range key.
testPutManySuccess	Confirms that PersistentStorage successfully appends many key,value pairs from a message with a PUT_MANY status.
testPutManyError	Confirms that PersistentStorage does not append many key,value pairs from a message with a NO_STATUS status.
testGetDataInRange	Confirms that PersistentStorage removes key,value pairs from storage that are within the given hash range.
testGetDataInRangeOnDeath	Confirms that PersistentStorage removes all key,value pairs from storage when the server must die.