

ECE419 Milestone 2

Wuqi Li 1000292033

Yuhui Pan 1001239395

Wuzhen Shan 1000411173

File System Design Document

Design Overview

The goal of this milestone is to extend the storage server on top of Milestone 1. The storage will be a controllable, scalable and data are stored over a number of storages servers with the help from consistent hashing and zookeeper.

General Responsibility Of Each Component

KVServer and KVClient works as the way they are in milestone 1 except that right now KVClient can access distributed data storage servers as a whole, KVServer will coordinate between each other to share data and communicate with the ECS to achieve the distribution purpose. ECS is the new component that we add to the entire system, it serves as a coordinator between server and server as well as itself and the servers.

Generally speaking, ECS take control over the entire data storage service, deciding on boost or shutdown over the storage servers.

Information carried from last milestone is as below (you can refer it by reading our previous report) :

Communication Logic I (CommunicationModule)

Translation Layer (KVStore & ClientConnection)

Cache system (KVCache)

File System (KVDB)

External Configuration Service (ECS)

ECS is basically made up of three parts: ECSCClient, ECS, ZookeeperWatcher. The interface part is ECSCClient which is responsible for providing an interface for Admin to manipulate the entire data storage servers' set-up. It directly talk to the class named ECS that we defined which serves as a bridge between ECSCClient and ZookeeperWatcher. The working mechanism between these three parts is as follows. ECS holds one ZookeeperWatcher instance which has an zookeeper client inside that is communicating with the zookeeper server that we manually start. We shift all the zookeeper related job to Zookeeperwatcher, it is responsible for setting up, deleting, watching nodes, and writing, retrieving data from nodes. ECS is translating all the messages from Admin side to the Zookeeper side as well as managing the server repository. The data structure we use to manipulate the servers is the ECSNode, which stores the related information of all the servers.

New Data Message

Treeset<IECNode> is what we use to store different kinds of server repository, including the available servers to ECS and dynamically changing alive server list (metadata) that is shared between ECS and KVServer as well as KVServer and KVClient. Our metadata includes the server name, ip address, port, hash range. We use Gson to serialize this message to string in order to make the transmission easier.

Communication Logic II (Zookeeper between ECS and KVServer)

Unlike what we did in milestone 1, we do not use direct TCP connection to transfer message between ECS and KVServer. Instead, we use Zookeeper to do the job. We have the following structure to complete the job.

Root Node (/ecs/):

The main task of the root node is to do broadcast for metadata update and is accessible from all nodes. All nodes will watch on this node for most-up-to-date metadata, which store all the available servers' list along with their corresponding hash ranges. The KV server will then forward the metadata as needed by KVClient. ECS will set up the root first and remotely launch different KVServers with required parameters for setup by ssh, and at the same time watch on potential standup of new child nodes (which we'll discuss below). Once KVServers are settled down, it will inform the ECS by creating new child node for further instruction but will remain as STOPPED state.

Child Node (/ecs/server[x]):

The majority operations will be done through child nodes including stop, shutdown servers, moving data between servers. The ECS will decide which servers to add as required by Admin, and then set corresponding child node data to make the transmission of data happen in order to match the new hash range requirement. Only the servers that are involved in the transmission will firstly the most-up-to-date information. After the transmission is done, the newly-added or newly-removed servers will inform the root node to broadcast new metadata to all servers. The principle is that whoever gets the WriteLock has the responsibility to inform the root of the change. Metadata can not be handed to other servers before the transmission is done. The secret is kept in involving parties.

Translation Layer II (KVStore & ClientConnection)

Unlike what we have implemented in Millstone 1, the data set has to be forwarded by client library to different, responsible storage servers.

Initialization: Client will connect to the server by user input firstly and set it as the default server. Initially, there will be no metadata available for client library. We assume that default server will either match the hash range as requested or provide the client with correct metadata.

Cache metadata of storage service:

The client will keep using the default server until it is provided with the metadata. And KVStore will store the metadata with the same format as it is in the ECS and KVServer, i.e. `Treeset<IECNode>`. The server will find the corresponding matched hash range there and start new connection if needed.

Metadata update and retry the request

Error handling

NOT_RESPONSIBLE :When clients send request to wrong server due to outdated metadata, the server will return an error message with latest metadata. Client will keep retrying to execute the operation until it reach the maximum retry which in our case is 5.

SERVER_WRITE_LOCK: When server returns this error message, the client will prompt the user for a later retry.

SERVER_STOPPED: When server returns this error message, the client will prompt the user for a later retry.

Storage Server (KVServer)









We modify some parts of the KVServer, but the structure remains the same as it is in Milestone 1. We also change some parameter settings in order to fulfill the requirement of Milestone 2.

And KVservers are initiated with an SSH call by ECS. However, the servers do not start accept requests from clients until told so by ECS. A message of "SERVER_STOPPED" will be sent back to clients.

Testing & Performance

The primary test framework we used is JUnit. We created 4 test classes(ECSClientTest, ECSNodeTest, KVServer and KVStoreTest, ConnectionTest, PerformanceTest and CacheTest). We have implemented all required testing functions as required in Milestone 2 handout. For example, we have tested initializing constructors for KVServer, check if KVstore is connected. For ECSNode, we checked if we can get node info by implementing `getNodeName()`, `getNodeHost()`, `getNodePort()`, `getHashRange()`. We also checked all the functionalities of add,setup,await and remove nodes by implementing corresponding functions as required in the handout. Moreover, we have have als tested KVStore by implementing test cases such as , get, put and disconnect,etc to test if KVStore works properly. A detailed screenshot of the test cases are in Appendix A. We also tested our performance using different cache size and strategies.

APPENDIX A

 AllTests	27	
 CacheTest	28	
 ConnectionTest	29	
 ECSCClientTest	30	
 ECSNodeTest	31	
 KVServerTest	32	
 KVStoreTest	33	
 PerformanceTest	34	
	35	
	36	
	37	

Test Cases