ECE419 Milestone 4

Wuqi Li 1000292033
Yuhui Pan 1001239395
Wuzhen Shan 1000411173

Design Overview

The goal of the last milestone is to extend External Configuration Service, KVServer and KVClient on top of Milestone 3. The storage service will be a controllable, scalable and robust entity with the backup from replicas setup by ECS. In addition to that, we have implemented a fresh Geographically-Aware Key-Value Store User System that enable clients to log into the system as different user such that their key-value domain can be isolated from each other, and their connection is always to the nearest server instead of reaching to the remote server that is holding the requested key-value pair. The main goal here is to noticeably improve the performance and robustness of the storage service and build up account system for better management of the Key-Value Store from the client perspective.

General Responsibility Of Each Component

ECS and KVServer has been modified altogether in order to cooperatively achieve the geographical functionality of Key-Value Store. ECS is still the brain of the system with the additional knowledge of geographical location of all servers and connected clients such that it is can always instruct the client to connect the nearest server as well as redirecting the traffic to the nearest server without notifying the client of the complicated mechanism implemented behind the scene. The KVServer has a lot of components added in such as another layer of cache for geographical purpose in order to make the performance boost feasible. The KVClient will not allow any operations until the client has log into the system as a registered user. All other functionalities such as crashed server detection, transmission of data between servers, etc., will all remain fully functional. The implementation details that is specific to this milestone will be fully explained in the following section.

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

Communication Logic I & II & III (CommunicationModule)
Translation Layer I & II (KVStore & ClientConnection)
Cache system (KVCache)
File System (KVDB)
External Configuration Service I && II (ECS)
New Data Message I & II (MetaData)
Note: For all the non-trivial details that has been achieved by Milestone 1,2 and 3, we just assume them to be known and may briefly mention them as necessary.

Translation Layer III (KVStore)

Geographic information are intensively involved and proceeded here. We enforce geographic information in all nodes and all client connection setup. The logic we use is to set x and y in every server and client to emulate the longitude and latitude of that entity on the earth and flatten them into a circle just like we did in consistent hashing. Every server and client will have a corresponding coordinates within the circle according to our setup. By such, we can easily compute the nearest server for every client and trigger the fasted speed connection. All the complicated logic like nearest server failure, response delay, etc. will be gracefully handled here in addition to what we have in Milestone 3.

KVServer

Of the KVServer side, it is responsible for connections between itself and all nearest clients. It is Also responsible for connections between itself and other servers that is initiated by the clients, since the client may put or get some key-value pair that is not sitting in this server, then this server has to reach out to the other server to do the operation on behalf of this client. An additional cache will be applied here in order to speed up response time. This additional cache is only serving the request that is not locally satisfied and has to reached out to the other remote servers. This means the server will cache some of the remote servers' key-value pair and serve them on behalf of the other servers. Besides, the additional cache is refreshed on a per request basis. This means "get" would become very fast if it hit a cache, and "put" will trigger an immediate response from the nearest server without waiting for the remote responsible server's acknowledgement. KVServer keeps a global state which it can refer to from Metadata it gets from ECS. All of these is benefited from leveraging the geographical setup and cache system. It helps reduce socket connections and latency of client having to reach the remote server by a long distance and restrict it within a shorter geographic bound. As our experiments come out, this improve the speed very much.

External Configuration Service III (ECS)

The principle we use for ECS is to involve ECS as little as possible, the only thing it adds on top of Milestone 3 is the geographical information embedding in the Metadata. The update and transmission of Metadata enable the server to be aware of the current status of the entire system and forward or receive traffic accordingly. ECS is not even aware of the user account system, all the isolation is handled in the KVServer and KVClient side. Just as Milestone 3, the zookeeper is coordinating all the communications including data transmission and you can refer to our previous report for detailed implementation of zookeeper instance.

User System

The strategy we use to create the user system is to disallow a client without a registered username and password. This is achieved by storing every single key-value pair by adding username into key, making it unique in the key-value store. When retrieving the key-value pair, the same strategy applied, and this is implemented in KVStore Layer which means the client will not be able to access other account data since the binding is enforced in the layer that is not accessible from the interface. And only the logged-in user is able to put and get within its own domain.

MetaData III

We add coordinates and a few methods on top of Metadata which facilitate the operations that is related to geographical information.

The following is a review the key components of Milestone 3:

Communication Logic III (Zookeeper between ECS and KVServer)

We have changed many parts of this module in order to fit the specific needs of this milestone as well as improving the efficiency and speed of Distributed Storage Service. The details will be covered as we go through how to we implement the following three sections:

1.Replication Mechanism Implementation

Since we load the ecs.config file entirely at the starting point of ECSClient, we are able to gain a complete hash graph of all servers before starting any server and it will remain relatively stable over the entire life cycle. We basically distribute the replica as instructed by the handout, the two servers right after. We use ECSWatcher to notify all servers of the incoming metadata along with the operation type all these servers has to comply to. Once the server gets the latest metadata, the server will immediately transfer the data to its replicas. The detail is that the server will call the utility functions inside MetaData to check its replicas and signal the target node of transferring action by creating new zookeeper node under that target node. All data are transferred by this zookeeper node and operations are guaranteed to be safe by acknowledging each successful transmission by target the node.

2. Failure Detection and Recovery

We have specifically design one additional class named ECSDetector to detect the failure. It is maintained as a Hashmap of ECSDetectors under ECS. The ECSDetector is basically simulating the behavior of an actual ECSClient by connecting to the KVServer. We have exactly the same number of ECSDetectors as that of running

KVServer instances. We connect each ECSDetector to each running KVServer, and once a server has crashed, it will signal the ECSDetector since there will be an exception thrown for a currently connected client if it gets disconnected by the other side, which in our case, is the ECSDetector that we are monitoring on. We have also registered ECS and the monitored node under every single ECSDetector such that when a failure is detected, ECSDetector is alway able to inform ECS of the event and passes back to crashed node. All the information can be retrieved from the node and ECS will remove it from the list and set up new node into the storage service from the available servers pool in order to maintain the server number. All the necessary data transmission will also be carried out as detection of this event and follows the same workflow of removenodes. But this time, this crashed server will not be put back into the reusable pool and ECS will never use it in the future.

3. Mechanism for clients to gracefully handle the failure of the storage server

Just like a when a client encounter a metadata change, once a client detect the connection session has failed, it will pull the first available server from the metadata and connect to it to see any change. And this is step may succeed or fail depends on the current storage service setup. If it fails it will get the latest metadata and through there it will find the right server to connect. If it succeed, it will remain as it is until the next time when it request something is not in the range of that server. Finally, it will get updated and sync to the latest metadata. There is only scenario that our client would fail, which is all the available servers in the metadata has crashed, and we are unable to handle this kind of failure.

Conclusion:

I think our product is mature and good with a lot careful design and considerations. We always bear in mind that the personal user service and convenience is the core of the product. The storage service is consistent with all previous milestones. And the location-service and user management we have in milestone is the silver bullet and will significantly enhance the user experience and functionality of this product. Last but not least, thanks so much for your hard work for reviewing and marking our milestone from 1 to 4. We learnt a lot from the milestone series and really enjoyed the project. :D