ECE419 Milestone 4 Design Document

Overview

In milestone 4, we mainly implemented a subscription notification feature that allows clients to receive updates when the values associated with certain keys have changed tackling different real-world scenarios. We based our M4 design on the original M3 design, which means that all subscription functions will have replicas and can handle server crashes.

In addition, we also implemented support for byzantine failures. With MACs mechanisms, we have encrypted all the communications between clients and servers with randomly generated secret keys. This can both protect the communication as well as mitigate attackers' unauthenticated messages.

Subscription and Unsubscription:

Before connecting to an available server, the client needs to specify its unique client ID as itself's identity. Then, a client can use the "sub" or "unsub" command to subscribe / unsubscribe to a series of keys (Figure 1). Each data in the server storage now has an extended attribute --- a subscription list to store the client IDs which have subscribed to this key. The subscription list data will be replicated to 2 other servers like any other normal key-value data. The storage data is basically in the form of Json:
key: {VALUE:value, SUBSCRIBERS:{client1, client2, client3}}

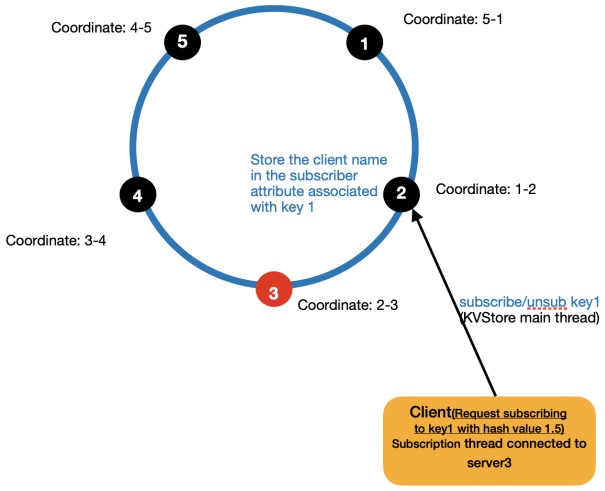

Figure 1 Client Subscribe to key1

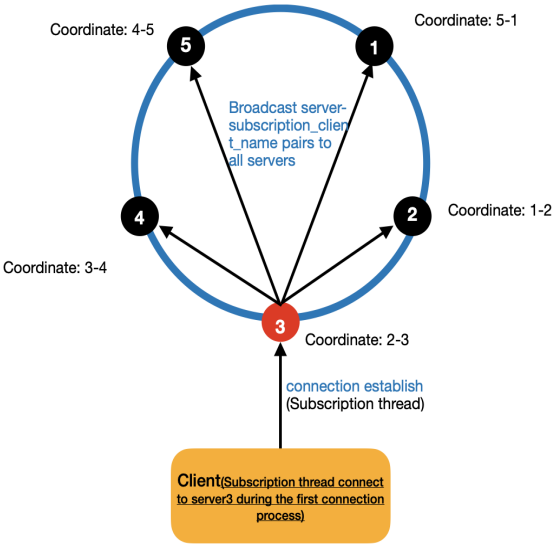

Figure 2 Subscription Thread Establish connection & Broadcast client-server connection info.

Upon connected to the server, the client will establish a separate thread with the server to receive notification messages from the server. Then, subscription relationship info will be broadcasted to all other servers (Figure 1). This tells the whole service that client A is currently connected to the service through Server B, which will be used to find the client to send back subscription updates after. The format of the data is like this: (client A - server B). All the info will be stored in a newly created data structure, called SubConnectionTable.

Notification

Once a server receives a key update that's in its range, the coordinator will get the key's subscribers and cross compare with the SubConnectionTable to determine the clients can be reached. Then the subUpdateToServers command will be sent to all the subscription servers connecting to the subscribers. (e.g, send to server B, to reach client A). Then, all subscription servers will check clientName-SubscriptionServer to get the subscription thread which is recorded during the first time the clients connected to the service. Then, the notification update info will be sent from subscription servers through the specific subscription threads to the responsible clients.

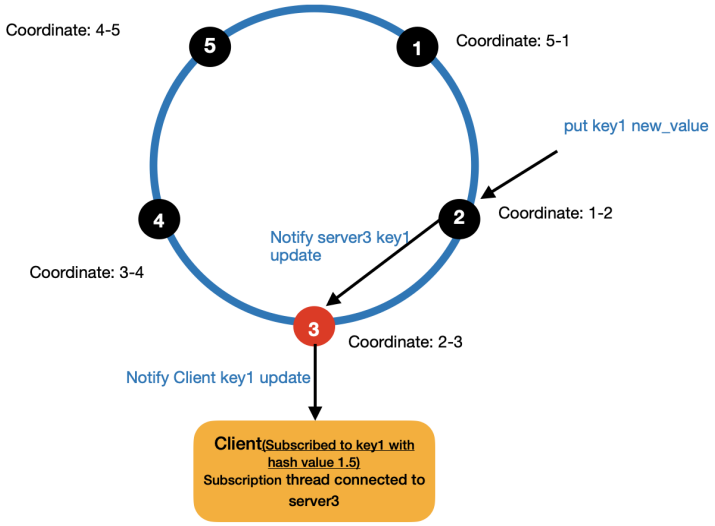

Figure 3 Key1 is updated by another client, notify the subscriber

Support for byzantine failures:

As figure 4 illustrates, everytime a connection is established, a client will first need to possess a password for authentication. After the client is authenticated by the server, the server will generate and send a one-time private key to the client. The private key is used to encrypt all the communications between client and servers in this connection session. Specifically, we applied the Advanced Encryption Standard

provided by Java " javax" module. Therefore, any malicious message that is not encrypted using the specific private key will not be readable and thus be ignored.

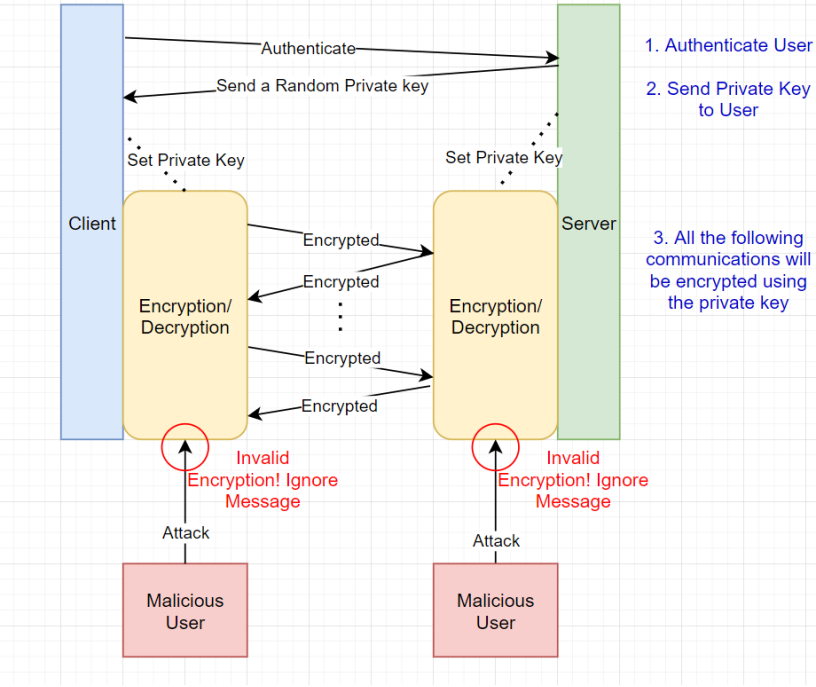

Figure 4 A flowchart of our byzantine failure support and encryption process.

Performance Report

The performance evaluation is conducted in the same way as Milestone 3. We measured its throughput and latency by issuing 800 PUT/GET commands at the ration of 50%:50%. Generally performance for all types of hash strategy have been lowered by 12-15% on average. (PUT request suffered slight greater performance loss)

The performance change could result from our extra features. During the PUT process, the server will now check an extra subscriber list to notify subscribers. This could cause PUT latency to increase. In addition, due to the encryption and decryption mechanism, extra steps are added between server-client communication, this could potentially lower the overall performance in server-client communication.

The performance of subscribe/unsubscribe commands is generally slightly slower than normal GET/PUT. This is due to these commands must wait until the server replicates the extra subscription list to replicas.

Appendix

FIFO Size 100	5Client / 5Sserver	5 Client 20 Server	5 Client 50 Server	20 Client/ 5 Server	20 Client/20 Server	20 Client/50 Serve
Request/sec	3.24675324675325	123.533045089561	160.128102481986	2.75785990071704	96.6183574879227	112.994350282486
Avg Latency (ms/request)	308	8.095	6.245	362.6	10.35	8.85
Avg Get Latency (ms/request)	262	13.8	4.55	366.2	9.4	8.5
Avg Put Latency (ms/request)	354	2.39	7.94	359	11.3	9.2

FIFO Size 500	5Client / 5Sserver	5 Client 20 Server	5 Client 50 Server	20 Client/ 5 Server	20 Client/20 Server	20 Client/50 Serve
Request/sec	3.95335046451868	47.5059382422803	145.985401459854	4.42086648983201	107.52688172043	129.032258064516
Avg Latency (ms/request)	252.95	21.05	6.85	226.2	9.3	7.75
Avg Get Latency (ms/request)	220.3	16.7	4.8	190.1	8.2	5.5
Avg Put Latency (ms/request)	285.6	25.4	8.9	262.3	10.4	10.0

LRU Size 100	**5Client / 5Sserver**	**5 Client 20 Server**	**5 Client 50 Server**	**20 Client/ 5 Server**	**20 Client/20 Server**	**20 Client/50 Serve**
Request/sec	3.25044693645376	45.5580865603645	132.450331125828	4.06231592630959	97.0873786407767	118.343195266272
Avg **Latency (ms**/request**)**	307.65	21.95	7.55	246.165	10.3	8.45
Avg Get Latency **(ms/request)**	284.0	17.0	5.4	200.83	8.9	5.1

Avg Put Latency (ms/request)	331.3	26.9	9.7	291.5	11.7	11.8

LRU Size 500	5Client / 5Sserver	5 Client 20 Server	5 Client 50 Server	20 Client/ 5 Server	20 Client/20 Server	20 Client/50 Serve
Request/sec	4.17536534446764	57.8034682080925	153.846153846154	4.07166123778502	105.820105820106	126.582278481013
Avg Latency (ms/request)	239.5	17.3	6.5	245.6	9.45	7.9
Avg Get Latency (ms/request)	201.1	10.2	4.9	242.5	7.0	6.2
Avg Put Latency (ms/request)	277.9	24.4	8.1	248.7	11.9	9.6

LFU **Size** 100	**5Client / 5Sserver**	**5 Client 20 Server**	**5 Client 50 Server**	**20 Client/ 5 Server**	**20 Client/20 Server**	**20 Client/50 Serve**
Request/sec	3.1201248049922	57.8034682080925	153.846153846154	4.07166123778502	105.820105820106	126.582278481013
Avg Latency (ms/request)	320.5	17.3	6.5	245.6	9.45	7.9
Avg Get Latency (ms/request)	291.8	10.2	4.9	242.5	7.0	6.2
Avg Put Latency (ms/request)	349.2	24.4	8.1	248.7	11.9	9.6

LFU Size 500	**5Client / 5Sserver**	**5 Client 20 Server**	**5 Client 50 Server**	**20 Client/ 5 Server**	**20 Client/20 Server**	**20 Client/50 Serve**
Request/sec	4.12031314379893	53.0503978779841	138.888888888889	3.89863547758285	93.8967136150235	111.731843575419
Avg Latency	242.7	18.85	7.2	256.5	10.65	8.95

(ms/request)						
Avg Get Latency (ms/request)	208.3	10.9	5.4	250.7	8.1	7.0
Avg Put Latency (ms/request)	277.1	26.8	9.0	262.3	13.2	10.9

Table 1. Performance comparison with different cache sizes, strategies, number of servers and clients

FIFO Size 100	5Client / 5Sserver	5 Client 20 Server	5 Client 50 Server	20 Client/ 5 Server	20 Client/20 Server	20 Client/50 Serve
Request/sec	3.14663310258024	32.626427406199	116.959064327485	3.41705108491372	87.3362445414847	106.951871657754
Avg Latency (ms/request)	317.8	30.65	8.55	292.65	11.45	9.35
Avg Subscription Latency (ms/request)	324.2	31.5	8.1	291.1	12.1	9.5
Avg Unsubscription Latency (ms/request)	311.4	29.8	9.0	294.2	10.8	9.2

Table 2. Subscription Performance with different number of servers and clients

Unit Tests

Test Name	testNewPutAndGet
Status	Pass
Description	Since in M4 we have changed our implementation of storage read and write (from CSV to Json using GSON), we want to make sure the original put and get functionality still works.

Test Name	testSubConnectionTable
Status	Pass
Description	In M4, we added new information to record a client and its connected server relationship on the server side. We want to make that this newly added data serves its expected functionality.

Test Name	testSubData
Status	Pass
Description	Testing the functionality of the newly added API for key subscription.

Test Name	testUnsubData
Status	Pass
Description	Testing the functionality of the newly added API for key unsubscription.

Test Name	testDidSubData
Status	Pass

Description	Testing the assisted functionality to verify whether the current client has already subscribed a specific key.

Test Name	testBroadcastSubConnectionTable
Status	Pass
Description	When a server receives the subConnectionTable, it will broadcast this updated info to all other available servers. We want to verify the process against our expectations.

Test Name	testSubReplicatedData
Status	Pass
Description	In M4, we still design to guarantee the subscription data replication in case of server crash. We want to verify this functionality.

Test Name	testUnsubReplicatedData
Status	Pass
Description	In M4, we still design to guarantee the unsubscription data replication in case of server crash. We want to verify this functionality.

Test Name	testSubscriptionThread
Status	Pass
Description	In M4, a client will initiate a subscription thread with a server to receive the subscription update besides the normal thread for sending commands. We want to verify such behavior.

Test Name	testSubUpdateFlow
Status	Pass
Description	This is a very comprehensive test, ranging from the server detecting the value update, sending value update info to all responsible subscription servers, and then all responsible subscription servers sending the value update to their connected clients. We want to verify this flow against our expectations.

Test Name	testAESEncryption
Status	Pass
Description	Testing the functionality AES Encryption unit.

Test Name	testByzantineFailuresToServers

Status	Pass
Description	Testing the byzantine failure support. Make sure the server does not stop when it receives random messages or messages from an unauthenticated source.