Milestone 5: Further Improvements in the Distributed-Replicated Key-Value Database

Cenk Gurbet
cenk.gurbet@tum.de
Technical University of Munich
Munich, Germany

Selen Uygun
selen.uygun@tum.de
Technical University of Munich
Munich, Germany

Yağmur Duman
yagmur.duman@tum.de
Technical University of Munich
Munich, Germany

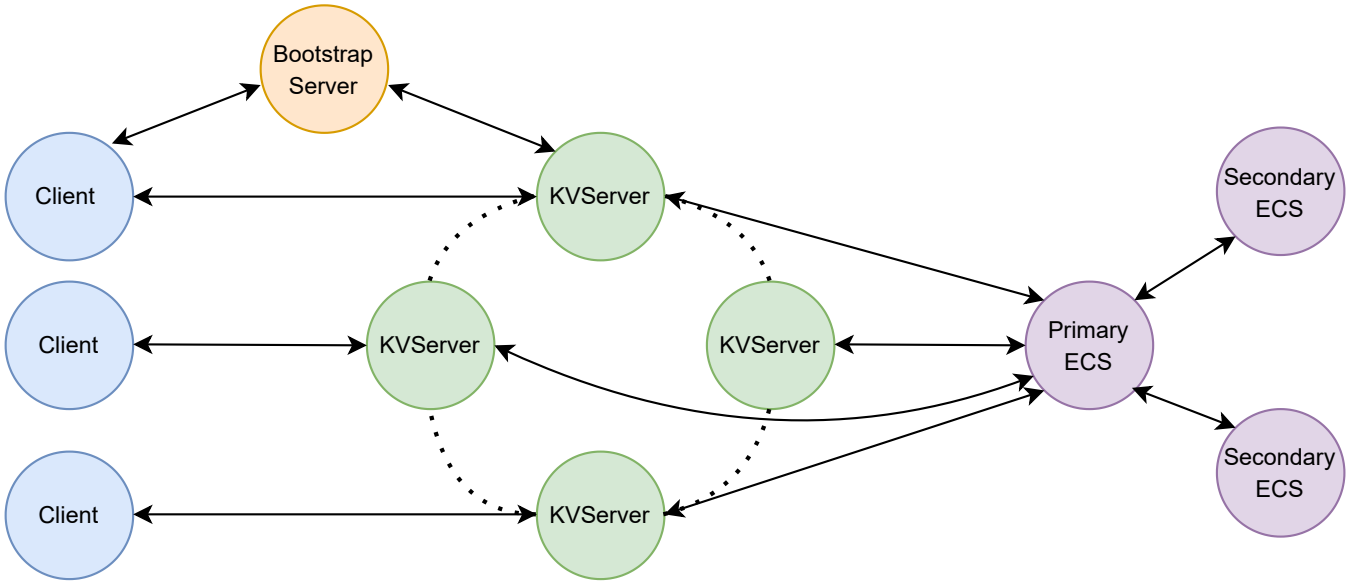

Figure 1: Overview of the System Architecture of Milestone 5

ABSTRACT

In today's world database systems are a huge component of almost every modern application architecture and implementing these systems in a distributed fashion provides a scalable and robust data management process. Throughout this paper, ways to improve the distributed and replicated key-value database implemented in Milestone 4 of the "Practical Course: Cloud Databases" will be discussed. Important topics such as addressing single points of failure, integrating encryption to the system for boosting security and efficient caching ways will be tackled.

CCS CONCEPTS

• **Computing Methodologies** → Distributed computing methodologies; • **Information Systems** → Data management systems; Information retrieval; • **Security and Privacy** → Database and storage security.

KEYWORDS

distributed systems, key-value databases, single point of failure, caching, encryption

ACM Reference Format:
Cenk Gurbet, Selen Uygun, and Yağmur Duman. 2018. Milestone 5: Further Improvements in the Distributed-Replicated Key-Value Database. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The need for robust, scalable and consistent distributed systems is omnipresent and one important applications of these distributed systems is in the concept of database systems. Distributed database systems are designed to store and manage data across multiple interconnected nodes while offering availability, fault tolerance, a better version of throughput, latency and scalability solutions to their single server counterparts. In the scope of this research paper we will go through the optimization of a distributed and replicated key-value store developed in Milestone 4 of the "Practical Course: Cloud Databases" offered at TUM.

This study will be focusing on several solutions for problems which are common in Distributed Storage Systems such as eliminating single points of failure by restructuring the External Server Configuration system, enhancing data retrieval efficiency by changing the data structure used for caching and finally addressing the security concerns through the integration of encryption mechanisms for client-server interactions.

The following sections will state the problems that were tackled and the approaches that were taken for the solutions in more detail while also shedding a light on the system architecture and design decisions that were made. Finally benchmark evaluation results where Milestone 4 is compared to the final version will be shared to show and discuss how certain design ideas affected the overall performance of the key-value database.

2 PROBLEM STATEMENT

The key-value structure we have taken over from MS4 faces critical challenges that affect its reliability and security when its compared to modern distributed applications. In order to overcome those and improve those deficiencies , several problems that are observed is going to be discussed in this section. As a team, we wanted to focus on the ACID properties that we believe that are significant for a distributed system to be concerned as "good". Additionally, rather than focusing on just one property to enhance, the team wanted to highlight various parts that are found as loose ends and wanted to be improve or enhance those parts. In our opinion, the main challenge with the structure that is taken over from MS4 was a single point of failure architecture in the system that is specific to the ECS server. This vulnerability exposes the entire system to downtime, data loss, and compromised data integrity in the event of an ECS server failure. Additionally there were other concerns that the group wanted to point out particularly on security and performance regarding data retrieval from the cache which will be discussed in this section.

2.1 Single Point of Failure

The challenge indicated above require immediate attention and comprehensive solutions to ensure the undeviating operation of the key value store system and the protection of client data. The extension of the MS4 aims to address the issues by simply adding innovative features to the existing key value store system. Its handled by introducing a coordinator server to resolve the single point of failure vulnerability inherited from MS4, and proposing an encrypted communication architecture by adding a Bootstrapper for more secure client-server communication.

The main goal while implementing the extension of the key value store system was to have an improved reliability and security in the key value store system. This goal is achieved by introducing a coordinator server and implementing secure encrypted communication between clients and the server. The coordinator server served as a mechanism that backs up the services provided by the External Configuration Server and ensures uninterrupted service. Encrypted communication between client and server ensures that the client of the system is more secure while using the key value store system we have provided, so that trust of the clients will be ensured.

To address the ECS server single point of failure vulnerability, a fault-tolerant system architecture that provides a back up mechanism must be designed. This vulnerability is handled by the team by having multiple External Configuration Servers and introducing a Primary ECS system which acts as a coordinator between ECS instances. Elimination of the "single point of failure" through this structure will ensure that the system will procure an uninterrupted service for its clients and an improved system availability. The availability of the system constitutes richer options for its clients by providing increased reliability. A client will be able to access the key value store system even when an ECS in the system fails as we always see in real world distributed system architectures. Additionally reliability property of the database systems is provided by implementing the back up mechanism.The system's detailed architecture on how its put into practice will be discussed in future sections.

2.2 Security

Encrypted communication between client and server communication will ensure the mitigating risks associated with sending the data in plain format. Since its found out that the possibility of data exposure in a database system is an huge concern, the team wanted to come up with an idea that is unique to us and provide a safer data exchange between client and server. Also we should have keep this in mind that current implementation sends the encrypted keys as in their format and the system would have evolved in a more sophisticated secure way in a certificate authority based approach for garnished security using Secure Socket Layers (SSL). Due to the need of providing a more secure platform for its end users in today's distributed system architectures; data exchanged between client and server in a plain format created a huge concern in our group. Thus, an additional encryption decryption system that provides a better platform for end users with a key generation is determined.By doing so data exchange between client and server would be in a more secure way. The encrypted communication framework based on key generation ensures data protection is enhanced keeping in mind that the requests coming from the client to the server side can have the possibility of containing sensitive data. The detailed architecture of how the security system is implemented is discussed in later sections.

2.3 Data Retrieval From Cache

Regarding the scalability of the database system, for large datasets, the performance was not enough and it had to be improved. For this purpose the most frequently used query in applications was considered which is the "get" query. Today, in most applications data retrieval is of utmost importance since most modern day application use this operation to display information, fetch data etc. The main goal with improving data retrieval performance was to prevent traversing the whole dataset for the get operation when the searched key existed in the cache, as this was unnecessary and had a worst time complexity of O(n). This O(n) time complexity made handling search operations in large datasets and big cache sizes harder as the amount of time it takes to retrieve the data from cache increased linearly and fast. In the long term this has to be avoided as databases are supposed to carry large amounts of data

and the purpose of cache is to improve performance by providing a fast look-up. To avoid this and utilize the cache better, cache was replaced with a map data structure rather than an array which reduced the look-up time complexity to O(1).

3 APPROACH AND SYSTEM ARCHITECTURE

3.1 Eliminating Single Point of Failure in ECS

Mainly we have prevented the Point of Failure issue for the External Configuration Server. In addition to that, newer version of our distributed storage system provides better cache utilization and secure communication between client and storage server. Firstly, we will describe how newer version handles point of failure of ECS system, and afterwards the secure communication and the new cache mechanism parts will be explained in more technical detail.

In a single ECS mechanism, all of the metadata information is stored in a single coordinator. So, if we have a problem/failure in that server we may lose all of the metadata information. To prevent that problem, we came up with a new approach for ECS. In our approach, we are getting multiple IP:PORT addresses from the user in the format of a configuration file. When user wants to start a new ECS, IP address and port information of the server to be added is checked to see whether it is an allowed address or not. Only if it is allowed then the new ECS is permitted to start.

We only have one Primary ECS at a time and multiple Secondary ECServers. Primary Server knows all of the other Secondary Servers' addresses (both IP and Ports), and also it has most up-to-date metadata information. Our new approach can be divided to two steps, these are: "Eventual Update of Secondary ECS Servers" and "Handling Failure in the Primary ECS".

For the "Eventual Update of Secondary ECS Servers", Secondary Servers regularly send "get" requests to Primary Server to get metadata/local view, but this approach is not strongly consistent. Secondaries eventually get metadata and update their local information. Request interval can be changed in the implementation however, increasing or decreasing the interval may cause problems. For example, if the get requests are sent very frequently this will increase the network traffic. Or if servers send requests with very little time intervals, they would most likely get out-dated quickly.

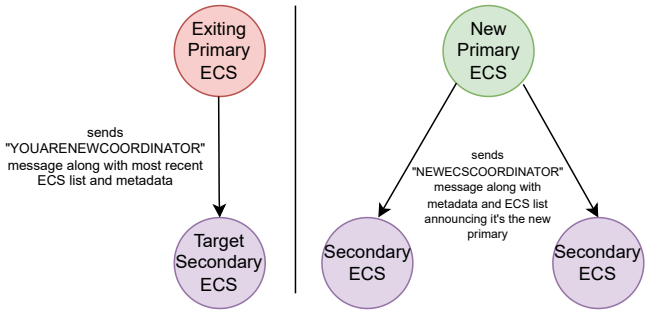

Figure 2: Approach to handling graceful exists of Primary ECS

Furthermore the second part of the newer version is "Handling Failure in Primary ECS". For this part, we follow different methods according to whether there was a graceful exit or not. If we have graceful exit, we utilize shutdown-hooks in termination. In the shutdown-hook, we select one of the Secondary Servers as the new candidate Primary Server. It is important to note that, as stated, Secondary Servers' metadata and local view for ECS Servers cluster is not the most up-to-date. So, firstly, before acting as a Primary ECServer, the Secondary server needs to be updated with the most recent values. So, in graceful exits, when the Primary ECServer is dying, it connects to the candidate Secondary and sends the most up-to-date metadata and local view information to that Secondary Server (local view is IP addresses and Ports of all of the ECServers in the cluster). Candidate Secondary Server can be selected randomly, but in most P2P systems (our new ECS approach can be seen as P2P system example), old servers generally are more secure than new ones (attackers can stay as innocent for a long time but generally old ones are considered more reliable). So, in our implementation we are selecting the oldest Secondary ECServer, which comes after the dying Primary ECServer in metadata, as the candidate server for the new Primary Coordinator ECServer.

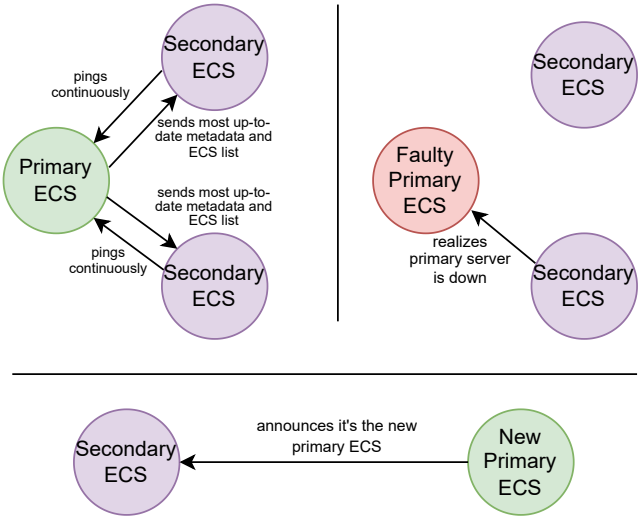

Figure 3: Three step approach to handling of unexpected failures in Primary ECS

If we do not have graceful exit, we need to follow a different approach as we can not handle all exiting situations of Primary ECServer at the shutdown-hook since a server can crash without entering there. To handle these kinds of situations we use a continuous ping mechanism. All of the Secondary ECServers send ping requests to the Primary ECServer. When one of the Secondary ECServers detect that the timeout for the ping request is reached (response did not come in expected time so most probably Primary ECServer is dead), it uses a synchronized method to become the new Primary Server. We are using a synchronized method because if we don't, all Secondary ECServers which detect the timeout can act as a new leader which would cause an ambiguity and we want to avoid this scenario. So, after detecting the dead Primary ECServer, Secondary ECServer sends an update to the rest of the ECServers listed under its metadata, announcing that it is the new Primary

ECServer, then the Secondaries update their coordinator ECS information. It also deletes the previous Primary ECServer which is assumed dead from its metadata and sends its metadata to the other Secondary ECServers as well so that they are up to date with the metadata of the most recent Primary ECServer. Since there is an eventual consistency in the updates of the metadata as stated previously, it can't be made sure that the new Primary ECServer had the same metadata as the previous Primary ECServer, however while taking this architectural decision the availability of the ECServers were prioritized over consistency.

3.2 Encryption of Client-Server Interactions

In initial version, we have an non-secure system in messaging between Clients and Servers; because, all data that has been used in transitions were not in an encrypted format(raw). Someone, e.g. attacker, can read/see data which was sent from Client/Server to another one. This may cause some problems such as confidential data might be seen by others. In order to solve that problem we added encryption mechanism via Encryption Bootstrapper.

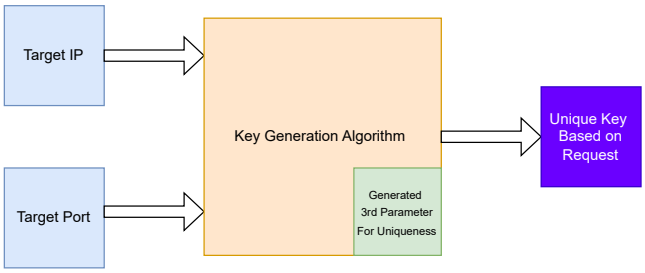

Figure 4: Key Generation Algorithm Structure

The figure above demonstrates the basic logic behind the bootstrapper's key generation algorithm. The bootstrapper primarily gets TargetIP and TargetPort information from its client. In our case it can be client or server. 3rd parameter is generated within the algorithm to create the uniqueness of the generated key. However in general to be more clear we can indicate that all of the Clients and Servers send 3 parts of information to sent to Bootstrapper before sending actual data to the destination and get Encryption Key from there. These information are "targetIP", "targetPort" and "thirdEncryptionKey". The value of "thirdEncryptionKey" is unique value which is calculated from "targetIP" and "targetPort" with our mathematical formula.

The formula above demonstrates the key generation algorithm's steps as it receives the IP number and Port number from its client based on the interaction it can be whether client and server. According to the need of the operation (encryption or decryption) the needed key is generated following those steps.And the above steps particulatly creates the 3rd parameter of the key generation algorithm.. Since the encryption algorithm that's generated is a symmetric algorithm that creates the similar keys for the mentioned operations, the type of operation does not affect how the key is generated particularly. Since we are heavily influenced by how Aes works , we should keep this in mind that according to the studies,

$$
\begin{aligned}
&(1) 127 \,\%\, 100000 = 127 \\
&(2) \quad 0 \quad \%\, 100000 = 0 \\
&(3) \quad 0 \quad \%\, 100000 = 0 \\
&(4) \quad 1 \quad \%\, 100000 = 1 \\
&(5) \; 34345 \; \%\; 100000 = 34345 \\
&\\
&(6) \; [\,(1)+(2)+(3)+(4)+(5)\,] \,\%\, 100000 = 34473 \\
&(7) \quad 100000 - 34473 = 65527
\end{aligned}
$$

Figure 5: Example of 3rd parameter creation steps for IP number 127.0.0.1 and Port 34345

number of operations based on secret key generations affects and reenforces the affect of AES algorithm.[1]

Formula calculates the unique "thirdEncryptionKey" from other 2 inputs based on the request of the client. We are using complex mathematical formula pointed above in here, and by doing so it will be challenging for an possible attacker (men-in-the-middle) to find out the exact 3 values at the same time. After getting "KEY" from Bootstrapper, we as in the place of Clients/Servers, encrypt the actual data with that "KEY" received from the bootstrapper so that has been generated according to the formula shown above and send this data to destination.

When target of the system which is Server/Client in our case, receives the packet/message from source. At the beginning target is not able to read data that has been sent since it is in the encrypted format. So, it primarily needs to get "KEY" firstly to conduct encryption process, in here, "KEY" is specific to the target address. The target can be either server or the client. Server/Client sends it's required information by the algorithm; "IP Adddress" and "PORT" values to Bootstrapper to get "KEY" to conduct the operation. After getting key value for decryption for instance, it can decrypt data and see what operation (put/get/delete) it is according to the result it receives. After doing actual operation, it also encrypts response with the same mechanism (Getting key from Bootstrapper, encryption of message) and send response back to source.

3.3 Improving the Performance of Data Retrieval from Cache

In terms of database caching the focus was on improving the time complexity of the data retrieval from cache for LRU and FIFO caching policies while the time complexities of insertion and deletion remained the same. To get a better understanding of the improvements that were made here, a brief overview of the overall caching structure of Milestone 4 will be shared in the following paragraph.

In Milestone 4 an array of key-value objects were used for the cache data structure, and when a get operation was performed, first the cache array was traversed to see if the key-value pair existed in the cache. If it did then the read operation from disk wasn't performed and the value was directly returned from the cache. While this caching way saved the users time, as the json file stored in the disk wasn't opened or read and the value for the corresponding key was directly returned, it still had a worst time complexity of O(n). This meant that there was a linear growth

in time as the database system grows. As the size of the dataset and the cache array grew so did the time it takes to retrieve the data from cache and in worst-case scenarios unwanted waiting times could occur leading to decreased application performance and responsiveness. In order to mitigate this unwanted behavior, and improve the system performance while also making the system more responsive for get operations, the data structure used for caching was changed.

As mentioned, previously an array was used for the cache but with Milestone 5 a FixedCapacityMap was introduced to substitute this array structure. This FixedCapacityMap extends LinkedHashMap class of Java which offers an efficient way of maintaining access-order in the map that is suitable for LRU and FIFO cache policies. Removing, adding and searching for an element in this data structure has O(1) time complexity. The downside is, for randomly accessing the elements, it has O(n) time complexity which is a downgrade from an array that does this in O(1) time but for the sake of speeding up the get operation this was disregarded. When a get operation is performed, first the cache is searched using LinkedHashMap's containsKey() method if it is found then returning its value to the user takes O(1) time. This method is very beneficial as the size of the cache and dataset increases the time it takes to retrieve the data stays the same as the time complexity for that operation is now O(1) constant time.

4 BENCHMARK EVALUATION

The benchmark evaluation can be gathered under three topics and in all of the benchmarks the Milestone 4 operations are compared to Milestone 5 operations. All of the tests were performed with a dataset that consists of 100 key-value pairs and using one ECS one KVServer and one Client for simplicity purposes and to focus on the get-put-delete operations solely. 10 queries were performed for each operation and the average of the latencies was taken into consideration.

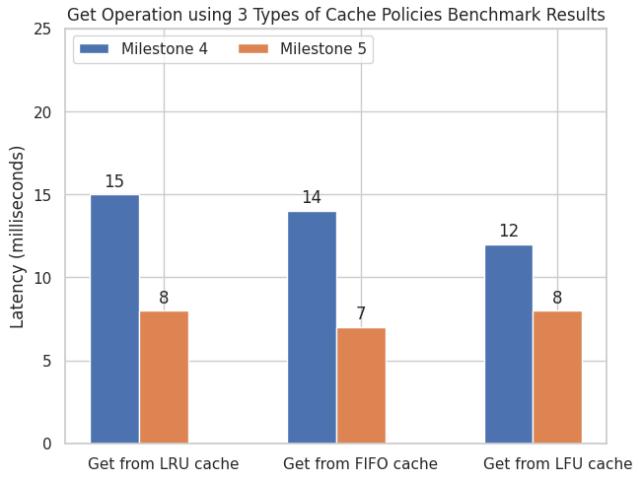

Figure 6: Get Operation Using 3 Different Cache Policies Benchmark Results

In Figure 6 the benchmark results are shown for retrieving data from cache for get operations in three different cache policy scenarios. To make sure the cache was utilized while retrieving the data, the cache size -the capacity- parameter was given as an integer value that was bigger than the size of the dataset so bigger than a 100. From the results it can be seen that for all of the caches, using the FixedCapacitySize map worked well as the latency decreased by a noticeable amount, this means that the time complexity for look-up operations improved by using a map instead of a cache.

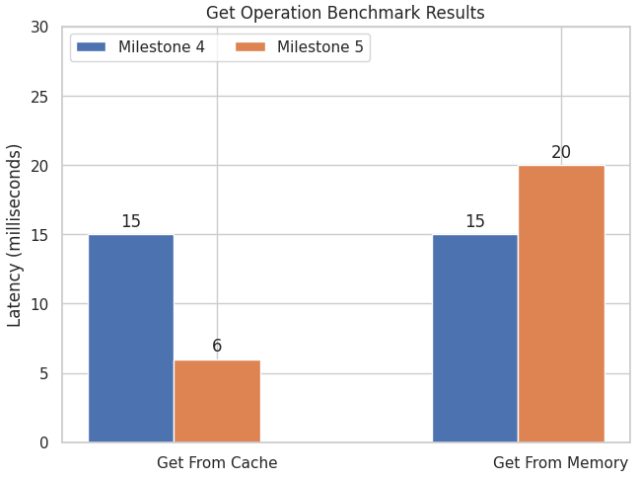

Figure 7: Get Operation Benchmark Results in MS4 and MS5

From the benchmark results seen on Figure 7 it can be derived that even though the time complexity of data retrieval from cache improved in Milestone 5 where the map data structure was used, the time complexities for data retrieval when keys don't exist in the cache stayed more or less the same. This behavior was expected as if the key didn't exist in the cache the memory had to be traversed which has a time complexity of O(n). Since the improvement was only on the cache data structure whereas the way memory is stored which is in the format of a json file didn't change, this meant that the json file still should be read and searched to find the requested key-value pair for a get operation. For get operations it's important the note that the frequency updates of key-value pairs are performed after a success message is returned to the user, to increase availability to the user and return fast responses.

Finally, the results of get and put operations were compared for both Milestones. The benchmark results can be seen on Figure 8. For the put operation it can be said that there was no improvement in terms of time complexity however this was also an expected behavior as the focus was on improving the latency for data retrieval -get- operations. For a put operation, since an update on the memory was required to return a success message to the user the json file had to be traversed which again has linear time complexity. Besides this, the memory had to be checked to see if the requested key-value pair existed in the database to determine if there was a put update scenario. Likewise for delete operation the memory needed to be traversed to find the element to delete and then the memory also had to be updated with the changes so that the requested key value

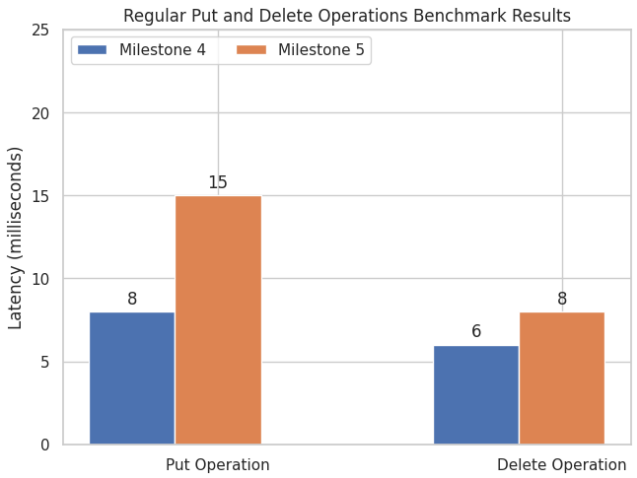

Figure 8: Put and Delete Operations Benchmark Results in MS4 and MS5

pair is deleted from the database and these operations required O(n) linear time complexity.

5 DISCUSSION

5.1 Single Point Of Failure

The external configuration server holds the data within an hash ring in order to connect the clients to the respective server. However in case of downtime of ECS, there would be an interrupt between client and server communication. To overcome this problem; As its mentioned in Architecture part, we developed PrimarySecondary mechanism to deal with that problem. But, overall system still prone to the failures such as electrical shortage for all servers at the same time etc. So, that can be developed further backup storage systems in addition to Secondary ECS Servers.

5.2 Improving Security

For the key generation part of the project,the purpose was implementing an encryption algorithm. With the additional calculation of the 3rd parameter; the uniqueness is created in the key generation part of the Bootstrapper Server. However its also not 100 percent secure that sending the encrypted version of the client requests (which are put/get/delete with the respective key) from both sides between client and server. It would have been better to have an SSL based approach with a certificate authority in which all the encryption-decryption process parameters won't be exposed in an encrypted format between client and server.But also it should have to be kept in mind that approach would not be in a unique way and the team tried to come up with a unique approach by using mod and XOR operation in key generation. The implementation details were discussed in the prior sections. Additionally while implementing the security concern of the MS5, one of the biggest issues that have been discussed among encryption algorithms is that not all encryption decryption algorithms are eligible to be used by all types of client devices. That was one of the main reasons of the choice of our own algorithm that's highly recommended by too many users. There should have been an encryption decryption approach that should be scalable and having the space to bring options for the developers to design their own algorithm .

6 CONCLUSION

As a conclusion, we developed our existing Distributed Key-Value storage system in various fields such as improved cache storing, handling ECS single-point-of-failure and secure messaging between Clients and Storage Servers. For the secure messaging part of the project; it has been known that SSL (Secure Sockets Layer) is a widely adopted encryption method due to security and compatibility level, the group wanted to come up their own idea to have a more secure transition between client and server communication by using their own algorithm. And combining encryption algorithm with a unique algorithm pointed out that secure messaging can be conducted between client and server by not only implementing an already known but in a way that combines various approaches.

REFERENCES

[1] Domaschka, J., Hauser, C. B., and Erb, B. Reliability and availability properties of distributed database systems. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference* (2014), IEEE.

[2] Zhou, J., Xu, M., Shraer, A., Namasivayam, B., Miller, A., Tschannen, E., Atherton, S., Beamon, A. J., Sears, R., Leach, J., Rosenthal, D., Dong, X., Wilson, W., Collins, B., Scherer, D., Grieser, A., Liu, Y., Moore, A., Muppana, B., Su, X., and Yadav, V. Foundationdb: A distributed unbundled transactional key value store. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)* (New York, NY, USA, 2021), Association for Computing Machinery, ACM, pp. 2653–2666.

[2] [1]