Praktikum: Cloud Data Bases
Final Report

Aly Kamel*
aly.kamel@tum.de
Technical University of Munich

Iremnur Kidil*
ge34hoz@tum.de
Technical University of Munich

Ricardo Kraft*
ga84vax@mytum.de
Technical University of Munich

ABSTRACT

In today's world, communication is key between people to retrieve information that is substantial. Almost every software system includes a messaging platform nowadays in order to ease their clients lives by offering a chat service where essential information can be shared. On this basis we extended our program with a group chat feature, where multiple clients can communicate with each other via chatrooms.

This report begins with a brief description of the replicated and distributed storage service from milestone 4. The replication strategy guarantees eventual consistency and basic availability, which is crucial for distributed database systems.

Furthermore, our extension is examined focusing mainly on how the system works from the client's point of view. Following that the features of the group chat and the implementation is studied in greater detail. The main idea is that multiple clients can join a chatroom with a chatID, assigned individually for every different room, and exchange messages there. Moreover, clients can perform read and write operations with the help of a chatbot while being in the chatroom.

Eventually we conclude our paper with a performance analysis by comparing results from our performance measurement tests and touch upon the advantages of providing a group chat.

CCS CONCEPTS

• **Distribution of data records(key-value pairs)** → **Storage service**; • **Replication** → *Redundancy*; • **Group chat** → Information retrieval system among clients;

KEYWORDS

client, server, key-value store, coordinator node, replica node, read-write requests, client application logic, external configuration service(ECS), replicated storage service, data distribution, consistent hashing, replication, eventual consistency, availability, data persistence, group chat, chatID, chatroom, userID, chatbot

*Authors contributed equally to this work.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Aly Kamel, Iremnur Kidil, and Ricardo Kraft. 2020. Praktikum: Cloud Data Bases Final Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Motivation–> TODO

2 BACKGROUND

One of the main goals implementing milestone 4 and milestone 5 is to have a system which is preferable for the clients by designing it with the following features: high availability, eventual consistency and partition tolerance.

2.1 CAP Theorem

According to the CAP Theorem, distributed database systems have three substantial properties to consider which stand for the name of the theorem itself: consistency, availability and partition tolerance. Eric Brewer, the founder of the CAP theorem, aimed to clarify with this theorem that a distributed database system can have at most two of the three mentioned properties. Reference: E. Brewer, "Towards Robust Distributed Systems

2.1.1 Consistency. When a system is focusing on consistency, clients should be provided with the most up-to-date data, meaning the fresh data that after the last write operation.

2.1.2 Availability. "A" in CAP referring to the property availability, centres upon responding the request of the client in any cases.

2.1.3 Partition tolerance. Partitions are failures which can be encountered in distributed database systems, namely crashed servers or dropped packets. Lastly, if a system maintains partition tolerance, the system will handle the problem without having to shut down.

Later on, Brewer mentions in his article "Cap 12 Years Later: How the Rules Have Changed" that designers do not have to abide strictly to the 2 of 3 principle, it is rather a spectrum than binary options. In other words, a distributed database system can favour high level of consistency and partition tolerance by having low level of availability. Thus, the initial theorem is improved by not having to sacrifice availability completely. Reference:Cap twelve years later

2.2 BASE

BASE is one of the design approaches in terms of the CAP theorem which is also created by Eric Brewer. Non-relational database systems(NoSql) make use of the BASE approach, which concentrates on high availability. BASE is an acronym which stands for basic availability, soft state and eventual consistency. With milestone 4, replication is added to the system with the intention of increasing the availability by distributing the data records to the two replica servers.

2.2.1 Basic Availability. Client is guaranteed to get a response from the servers in a matter of time, but the returned value may be stale, stated in other words not consistent, since the system is focusing on high availability.

2.2.2 Soft State. Without any updates state changes can be observed within the system.

2.2.3 Eventual Consistency. If there are no updates in the system for a long time, then all servers will gradually become consistent. reference: Milestone4 slides

Implementing the program, we paid attention that the system we created, provided the above-mentioned properties on a high scale. Our system contains the first property of BASE being basic availability, because servers will respond to request of the clients even if a latency occurs. Since single failing nodes can be tolerated due to replication, partition tolerance is provided as well. At last, eventual consistency is achieved within the system. If there are no updates for a long time all the replicas will become eventually consistent.

3 GROUPCHAT

The developed system is based on a client/server architecture which uses a many to many relationship. The program enables multiple clients to communicate with multiple servers. Besides replicated and distributed storage service, with the created extension clients are now able to exchange messages with each other.
In the following sections the functionalities of the groupchat is described by first representing newly added commands to the client library and then explaining the execution of the workflow.
The main goal here is to clarify the usage of the system for a client, who does not have the full knowledge of how a distributed database system works.

3.1 New Commands for the Groupchat

In addition to already implemented commands, such as "put" or "get" with the purpose of accessing the database or commands like "logLevel" for changing the level of the logs dynamically, 7 more commands are implemented in order to realise the groupchat extension.
Following commands are possible during a chat session:

(1) PUT <key> <value>:
Stores the given value and allows future access to it through the provided key.
(2) PUT <key>:
Deletes the value assigned to the given key.
(3) GET<key>:
Inserts the value assigned to the given key into the message.
(4) WSP <user1>,..,<userN> <msg>:
Sends a whisper to the users provided by the client. It is possible for a chatroom to have up to 30 clients in it. The logic behind whispering feature is a client should be able to send messages during a chat session, only to the people who he/she wants to share it with.
(5) QUIT:
Leaves the chat session.
(6) ACTIVE:
Returns a list of all users in the chatroom. A client does not have to always keep up with the notifications about who joined the chat or left it, that is why the command "ACTIVE" eases for a client to see online members at that moment.
(7) HELP:
Displays the help text.

Put and delete operations are fulfilled with the help of a chatbot. Most of the software systems include a chatbot in their system to work as a costumer service. Our intention by implementing a chatbot is to decrease the workload of the chatroom.

3.2 Chatbot

One of the extensions followed by the groupchat is a chatbot which is designed to help the chatrooms to access the distributed database. The chatbot shares a lot of similarities with the client application. When a client wants to perform read or write operations while chatting, requests are directed to the chatbot.

3.3 Execution of the Workflow

To start off, in the same sense as milestone 4, initially the External Configuration Service (ECS), where the storage servers are monitored and controlled, is executed. Following that, depending on client's decision, a number of servers are created. A client must connect to one of the servers by typing it's IP address and port number in order to use the database system.

3.3.1 Unique Username. After successfully connecting to a key-value server, the client has to either enter a username or use the command QUIT to have a username randomly assigned to him. Usernames are implemented as globally unique identifiers for the clients, which prevents different clients from having the same username in different chatrooms. That way users are guaranteed to know who they are communicating with as long as their partner is connected to the system.

3.3.2 Chat Command. Moreover, to use the chatroom client needs to type "chat" on the console and choose a chatID for the chatroom and type it right next to the command "chat". A chatroom is created with the given id, subsequently client have two options: either enter a private room with a password feature or a public room.

3.3.3 Private/Public Chatrooms. If the client is the first person to create a private chatroom, then the right to give a password to the room belongs to the same person.

3.3.4 Password. Another client who is connected to the same server and wants to chat in the same chatroom can only access to the private room with the selected password by the client, who created the private chatroom. The password is hashed in order to provide safety for the client. If client wants to have a public chatroom, then a password is not needed.

To prevent heavy workload for a server, a chatroom has the maximum capacity of 30 people. The chatroom offers a communication platform for all the clients sharing the same chatroom. Every message sent by the clients have timestamps in order to keep on track with the flow of the messages for other clients. One message can contain maximum 200 characters.
When a client joins a chatroom, all the messages which sent until that time, will be visible to the latest joined client and all the members will be informed about who joined or left the chatroom.

3.3.5 Saved Messages. The messages are saved into a .txt file under the directory of the respective server.

Whenever a client wants to leave the chatroom, QUIT can be typed in order to use the functionalities of replicated and distributed database from milestone 4. To enable the chatroom service client only needs to enter "chat" command with the desired chatID.

4 IMPLEMENTATION

In this section the implementation of the chatroom is examined in depth. Firstly, the extension for the client side and then for the server side are described respectively.

4.1 Client Side Implementation

Figure 1: Client side architecture

4.1.1 Client Library. The client side consists of three main components. The ClientApp represents the client interface which allows input through the console. From there the client is able to issue commands to: connect to and disconnect from the system, interact with the key-value store and chat. The input is then checked and parsed before getting sent to the ClientLibrary.

The ClientLibrary serves as a bridge between the client and server.

ActiveConnection abstracts the TCP socket connection to the server which allows the client to connect to the server socket and exchange data without worrying about the underlying structure.

4.2 Server Side Implementation

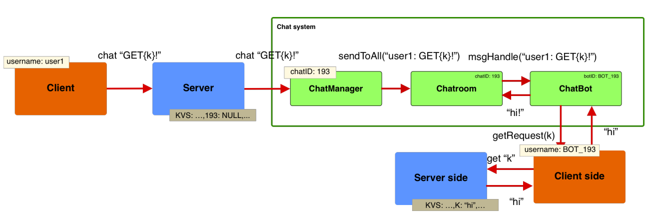

Figure 2: Illustration of a client sending a chat message

Each server owns a list containing the active chatrooms that it is responsible for. In order for a client to join one of those chatrooms it would first need to connect to that server, similar to the way storing a key-value pair functions. The decision to make chatrooms accessible only from one server has both its advantages and disadvantages. For one, it reduces the complexity of the system because otherwise servers would need a way to exchange updates regarding the active chatrooms and chat users whenever a user joins or leaves.

A problem with our implementation is that if the chatIDs are not equally distributed across all servers, which may occur due to the unpredictable nature of the hashing function, a single server could then be in charge of most chatrooms. This would cause that server to be overloaded with requests and would lead to greater response times and, in the worst case scenario, would result in a bottleneck for the whole system. In order to combat this issue, we limit the amount of chatrooms belonging to one server to 15 and the amount of users in a single chatroom to 30. This means a server is responsible for up to 450 chat users. These limits could also be easily changed depending on the intended use case of the system.

The biggest advantage of our decision is that it heavily reduces network traffic. Since all chatroom users are connected to the same server, that server can easily forward messages between them. Otherwise additional socket connections would have been required which would have both increased the load on the network and the overall complexity of the system.

Our idea for the chatting functionality was for it to be as lightweight as possible with clients entering and leaving chatrooms regularly.

5 SUMMARY

To conclude our paper, we

REFERENCES