CSE 512 - Distributed Database Systems Portfolio Report

Shreyas Chandrashekhar Kirtane
ASU ID – 1225453736
skirtan*1@asu.edu*

***Abstract*—Distributed Database Systems have emerged as a pivotal solution to manage vast amounts of data in contemporary computing environments. These systems distributed data across multiple nodes or sites providing advantages in performance and scalability. This document is a portfolio report for the Course Project of Distributed database systems undertaken in the semester of Fall 2023. It describes the application of a distributed database system for use in a complete design of a trading platform. It further describes the implementation of the solution in detail and the results obtained. Finally, it describes the lessons learned from the project.**

***Keywords*—Distributed database systems, Performance, Scalability, Trading platform design**

I. Introduction

Distributed database systems provide innumerable benefits when it comes to the modern computing landscape. Such systems have revolutionized the way data is stored, processed, and accessed. These systems excel in handling growing workloads. They can scale capacity by adding more nodes to the system and distributing the load smartly among these nodes. This improves performance. Since the data is stored across multiple nodes, it improves the overall reliability of the system. In the scenario that one node fails, other nodes with the same copy of data are available and can be used to serve user requests without any downtime. This technique of storing the same copies of data on different nodes is called replication. Not only does it improve reliability but also availability. Data can be replicated across multiple regions and data centers to provide fast access to data, improving latency and use experience. Advanced systems built on these principles implement complex protocols to ensure the data stored across multiple nodes is consistent and available for real-time analysis. This way of storing data is also cost effective, instead of building machines with more processing capabilities, it is significantly cheaper to distribute processing across multiple cheaper commodity machines.

These features play a vital role in shaping the financial sector, especially with the design of trading systems. Trading systems are required to process large amounts of data, ensure the system is available to all users maintaining fairness meanwhile maintain low latency. The use of distributed databases to build such a system could provide many advantages. To demonstrate these advantages, we have built a trading platform from group up using a New-SQL distributed database system. This project involves the study of various distributed database systems to select the system whose features are apt for the domain selected. We have provided a clear plan of action to implement the system, the intended task to be performed in each part, the tools used and expected deliverables. First, we go over the reasons for selecting the intended database system for the use case. Then we describe the implementation of the trading platform on the system. We describe the use of fragmentation and replication techniques used. We have also optimized some queries to demonstrate the use of query processing techniques. We then cover how transactions are handled in such a distributed system. And finally, we also implement the trading platform system on another No-SQL database to contrast it with the implementation in this project.

II. Description of Solution

The project used Cockroach DB [1] to implement the trading platform. Modern trading platforms have processing workloads which are increasingly geo-distributed. To handle such workloads Cockroach DB came out as the database of choice. It is a commercial database designed to support fault tolerance, high availability, geo distributed partitioning, replica placement and high-performance transactions. Cockroach DB (abbrev. CRDB) maintains at least 3 replicas of every partition in the database across geographic zones. CRDB is horizontally scalable, automatically increasing capacity and migrating as new nodes are added. It also implements a novel transaction protocol which supports performant geo-distributed transactions that can span multiple partitions. It supports serializable isolation and can be run on commodity servers, including those on public and private clouds. Now the solution to each part is described:

A. Design and Implementation of the distributed database system

In this section, we describe the schema design of the trading system and how the system is brought up on cockroach db. managed instance. To demonstrate all the capabilities of the database, we have used a managed cluster which involves servers located across 3 zones hosted on google cloud platform. Each zone has 3 nodes which run instances of the database. Cockroach DB is a NewSQL database. These solutions are based on the relational model. i.e. offer clients a pure relational view of data Even though clients interact with these data stores in terms of tables and relations, these solutions might use

different data representations internally [2]. So naturally, the trading platform follows a relational data model. The trading platform design has six different tables. User table stores basic user information related to each user. Accounts table keeps track of balance in the account and account type. It has a foreign key relation to the user table. Since, each user can hold only a single account. We have a transactions table to keep track of all the transactions that are executed on the trading platform. A stocks table which keeps track of all the stocks which are being traded on the platform, as well as a portfolio table which keeps track of all the different stocks and their quantities a user currently holds. An order table is created to store information related to orders a user makes on the trading platform, i.e. buy, and sell orders for stocks.

Since Cockroach DB provides a relational view of the data and is complaint with PostgreSQL. We decided to use the pgsql library with the python programming language to programmatically create these tables and foreign key relationships on the infrastructure. We used generative AI to produce mock data and stored the same on the database. We then used this system to demonstrate the abilities of cockroach DB and usage of distributed database system on the trading platform use case.

B. Fragmentation and replication techniques

The replication layer of Cockroach DB's architecture copies data between nodes and ensures consistency between these copies by implementing a consensus algorithm. It uses Raft [3], a consensus protocol which makes sure that data is safely stored on multiple machines, and that those machines agree on the current state, even if some get temporarily disconnected.

Horizontal fragmentation in a database involves splitting a single table's rows into multiple fragments based on a certain criterion [4]. To demonstrate this technique, we have created a table named frag_transactions from the transactions table. The table is partitioned by the transaction type. i.e. we divide the table based on if the transaction stored is a buy or sell transaction. Buy transactions are stored on east fragment and sell transactions are stored on the west fragment. We configured geo-partitioning feature so that the east and west partitions are stored in geographically different zones. This was possible since our setup consisted of 3 different zones. Using this database, we can also configure the distribution based on address stored on the user table or some ad hoc regex query on a particular field. We can also configure range partitioning to store transactions based on data in different locations.

Vertical fragmentation involves dividing a single table into multiple fragments vertically, by columns or attributed [4]. Each fragment contains a subset of columns from the original table. To demonstrate this, we divided the user table to store basic information which is also accessed frequently like user_id, name and city to a different fragment and less frequently accessed sensitive information like password at another fragment. We created these two fragments from the user table and stored them at different locations.

C. Query optimization

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage. Indexes are used to quickly find data without having to do scans over every row in the table. This technique is used in a distributed environment in this project. To demonstrate we execute a query on the transaction table for a certain account_id and between some transaction dates. We recorded the time it took to execute this query. Now we create an index on the fields (account_id, transaction_date) and try to run the same query again. We then compare the results.

D. Distributed transaction management

A transaction bundles several SQL statements into a single all or nothing transaction. Each transaction guarantees ACID semantics spanning arbitrary tables and rows. Transactions are atomic, i.e. if a transaction fails, nothing has changed on the database, whereas if it succeeds all the statements are executed virtually simultaneously. To demonstrate the capability of cockroach DB with distributed atomic transaction. We programmatically create multiple transactions in a multi-threaded environment. Also, these transactions are executed in different zones of the cluster. Since we have a cluster with 3 different geographic zones. We execute the same transaction in zone 1 and zone 2. Since this transaction tries to modify the same data, one of the transactions should fail. This scenario is used to demonstrate the distributed transaction processing provided by cockroach DB. Cockroach DB also provides semantics to implement user defined transactions. It provides isolation and rollback features on transactions. We have created these functions to demonstrate these techniques programmatically using python.

E. Distributed NoSQL Database System Implementation

We use MongoDB to build the same trading platform. MongoDB is a document database, i.e. a key-value database which stores documents or data in BSON format [2]. This provides flexibility in terms of schema. MongoDB also has an extremely rich query language, with real time data analysis capabilities. We host the MongoDB instance on a docker container and connect to it programmatically using the pymongo library for the python programming language. We create functions to perform operations to insert data into the database like trade samples, stocks, users, and market data. We implement code to retrieve this data and show all the collections on the database instance. Functions to simulate changes to market data and trades on stocks have also been implemented. This concludes the implementation of the trading system on a NoSQL database.

III. RESULTS

The script for each part can be run from the terminal using the command line. We were successfully able to deploy a

cluster which comprised of 3 zones with 9 nodes in total. We created and the trading platform and were able to simulate transactions on the same. After inserting data on any table from a single zone made it available across all the zones and nodes with minimum latency. With the use of fragmentation techniques, we simulated queries from different zones by deploying a server on the cloud platform closer to the data. Server in the west zone was able to quickly retrieve sell transaction data than any other server in a different zone. Similar results were found for the buy transaction type. Using

The query optimization part created an index to demonstrate the optimization. Using this we were able to reduce the latency for that query by about 50% due to the index on transaction date. The non-optimized version took around 0.083 seconds while the optimized version with the index took around 0.04 seconds.

With distributed transaction mechanism implemented in Cockroach DB were able to isolate transactions running on the same data across different zones. Same transaction run on different zones resulted in one of the transaction aborting due to conflicting changes on the same data.

We also implemented the same system in MongoDB. Although many of the features which come with Cockroach DB are absent, MongoDB shines with its query language resulting in intuitive queries for efficient data analysis and retrieval.

Thus, these results highlight the efficiency of distributed database systems like Cockroach DB. The problems relating to consistency, availability, and partitioning (CAP) have been around for a long time. Cockroach DB makes dynamic tradeoffs between these to provide best performance to applications such as the trading platform.

IV. My Contributions

A. Schema Design and Database software selection

I was responsible for coming up with a schema design which can satisfy the requirements of a trading platform system. I created the relational model for the use case and communicated with the group to include any improvements. I understood the problem statement, communicated with the TA and professor to clarify and ambiguities. I compared other systems like Volt DB other than Cockroach DB to make a pros and cons comparison.

B. Setting up database cluster and providing access.

After the database to be used was finalized. I researched the version of the database which was useful for the case. I set up the cluster with 3 zones and 9 nodes on google cloud platform. And created tokens for API access to the database to other users.

C. Replication and fragmentation techniques

To demonstrate the use of fragmentation techniques, I researched the fragmentation options available with CockroachDB. I also studied how fragmentation can be used in the context of the trading platform use case. I implemented the code to create the fragmentated tables and query them to retrieve the data. To test them I created Linux servers in different zones and noted the query latency, showing the advantage of horizontal fragmentation.

D. NoSQL distributed database system implementation.

I helped with the implementation of trading platform on the selected NoSQL database i.e. MongoDB. I implemented the code for creation of collections and insertion of mock data into the collection.

E. Final Presentation and Report

Created the presentation and report for the deployment of the solution and replication and fragmentation module. Also explained the solution in the demo video for the project.

V. Lessons Learned

This project has enriched my learning both in Distributed database systems and design of applications like the trading platform. I learned about the various advantages which distributed database systems provide. I learned about the tradeoffs these systems make. I also learned management techniques in addition to technical knowledge.

1. CockroachDB: Implementing the platform requires a thorough analysis of the underlying database to be used. What features did it provide and the shortcomings it has. I learned about setting up a cluster for CockroachDB as well as the API which can be used to store, retrieve data, and conduct transactions on the database.
2. MongoDB: I learned about the innovative data model provided by the mongo database. I also learned about the advantages of using mongo and the flexibility it provides with the data model. I learned what kind of data and relationships are fit for the data model used in mongo. I also learned about docker, to deploy an instance of MongoDB for local testing.
3. Trading platform system design: This project taught me the basics of a system which can conduct trades, with users, stocks, and transactions.
4. Teamwork skills: This project required a close collaboration of 4 members throughout the semester with varying mindset, schedule and thinking. I learned about the challenges of working with a team with independent contributions. Helping when team members get stuck and getting required help when I got stuck.
5. Python: I also improved programming knowledge in python with the use of various libraries like pgsql and pymongo to programmatically access database instances.
6. Raft: To understand how these database systems provide replication, I conducted a study of the raft consensus algorithm through paper in the reference [3].

VI. Team Members

The project had 4 team members in it. Other members are Aryan Mehta, Anuranjan Dubey and Mario Jones Vimal Stephen Antony.

References

[1] Rebecca Taft, Irfan Sharif, Andrei Matei, Natahan VanBenschoten, Jordan Lewis: CockroachDB: The resilient Geo-Distributed SQL Database, Sigmoid 2020, Cockroach Labs, Inc.

[2] Grolinger et al. Journal of Cloud Computing: Data management in cloud environments: NoSQL and NewSQL data stores

[3] Diego Ongaro and John Ousterhout, Stanford University: In Search of an Understandable Consensus Algorithm.

[4] Al-Sanhani, Asma & Hamdan, Amira & Al-Thaher, Ali & Al Dahoud, Ali. (2017). A comparative analysis of data fragmentation in distributed database. 724-729. 10.1109/ICITECH.2017.8079934.

[5] Daren Bieniek; Randy Dess; Mike Hotek; Javier Loria; Adam Machanic; Antonio Soto; Adolfo Wiernik (January 2006). "Chapter 4: Creating Indices". *SQL Server 2005 Implementation and Management*. Microsoft Press.