Project Presentation
CS744 Autumn 2024

A Distributed Key Value Store

Vishal Kumar Singh (24M0742)
Abhinandan Kumar (24M0788)

Context

What is the area/domain of the project?

The area/domain of the project is very vast it can be used as a foundations of designing any distributed system. The domain of this project includes

1. Database Systems
2. File Storage System
3. Content Delivery Network

Problem description

Statement:A distributed key-value store with different cache coherency and prefetching mechanisms

- A key-value store provides a data storage interface that uniquely references data items (objects) via a ***key***. Implement a key-value platform that maintains the data store in a distributed manner (split across multiple systems).
- Objects in the store can be stored and accessed from anywhere in the network, e.g., if there are 5 network-connected nodes in a cluster, each of them can add/delete/lookup the key-value store independently.

Scope/ Goal:

To create an efficient and robust system to efficiently store large number of key value pairs

Components of the project

List of main design and implementation building blocks

- Server (peer to peer)
- Database

1. Server

- **Consistent Hashing** :
  It ensures consistent hashing to perform storage and retrieval operations. This method allows node to dynamically add and remove without worrying about changing the hash function and costly data movements.

- **Provides Distributed Operations & Communication**:

  It provide communication facilities between multiple server to maintain data consistency and also ensure that all the server have the same up-to-date data.

2. Database

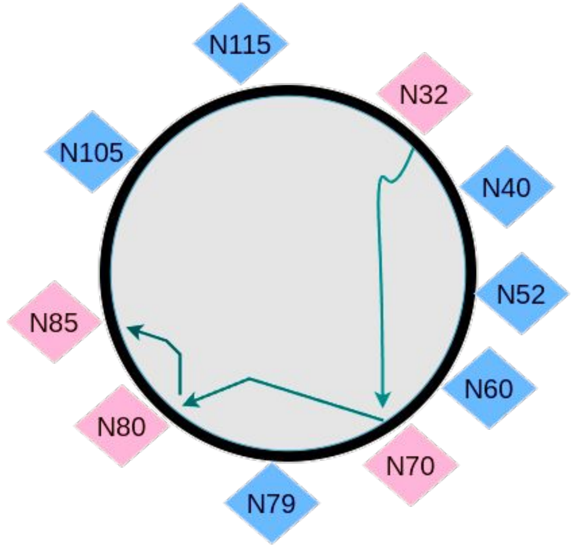
- **JSON-Based Persistent Storage**:

  The database stores data in form of JSON format, it helps to save and retrieve key-value pairs. This storage is persistent and the data remains same after the system restart.

- **The Server allow basic operations** :
  - **Create**: Add new key-value pairs.
  - **Search**: Fetch existing key-value pairs.
  - **Update**: Modify the values of existing keys.
  - **Delete**: Remove key-value pairs from the database.

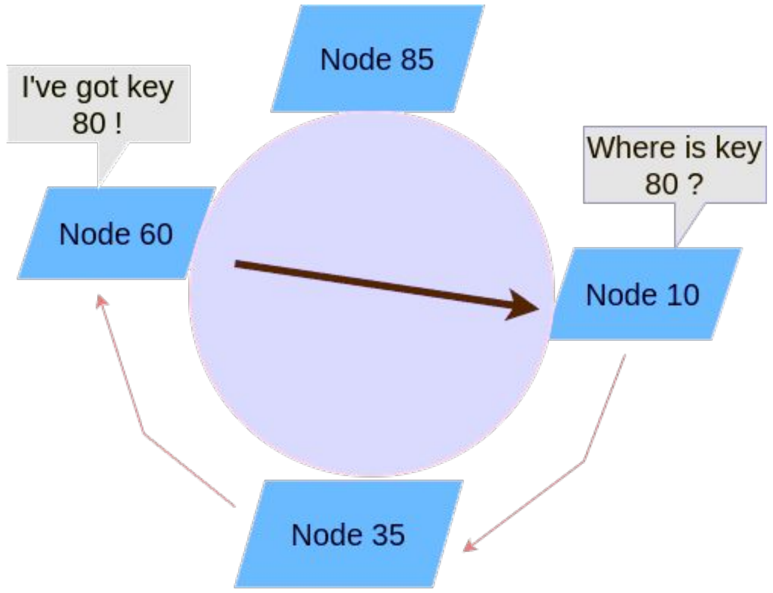
Design

Logical description of design aspects (use as many slides)
– components, interactions, actions, state, flow, examples, heuristics, decision choices, trade offs …



Design

- New node will ask position in the ring then it will join on that position
- For storing the data it will store on specific node and it successor until replication factor.
- For deletion of data it will delete on that specific node then it also delete from its successor.



Implementation details

1. Setup the New Node

- New node will find the hash value by their address.
- Then it request for the successor and predecessor from known Node, and will join the network on that specific hash position.

2. Store Key Value

- The node will firstly hash the key and it will find the appropriate node for storing the key value pair.
- It will also store the data on the successor node.

.3. Database Storage

- Data is stored in a JSON file.
- The database is simple and it easy to manage key-value pairs.
- Each server has their own database file that ensuring data isolation between servers.

.

Implementation details

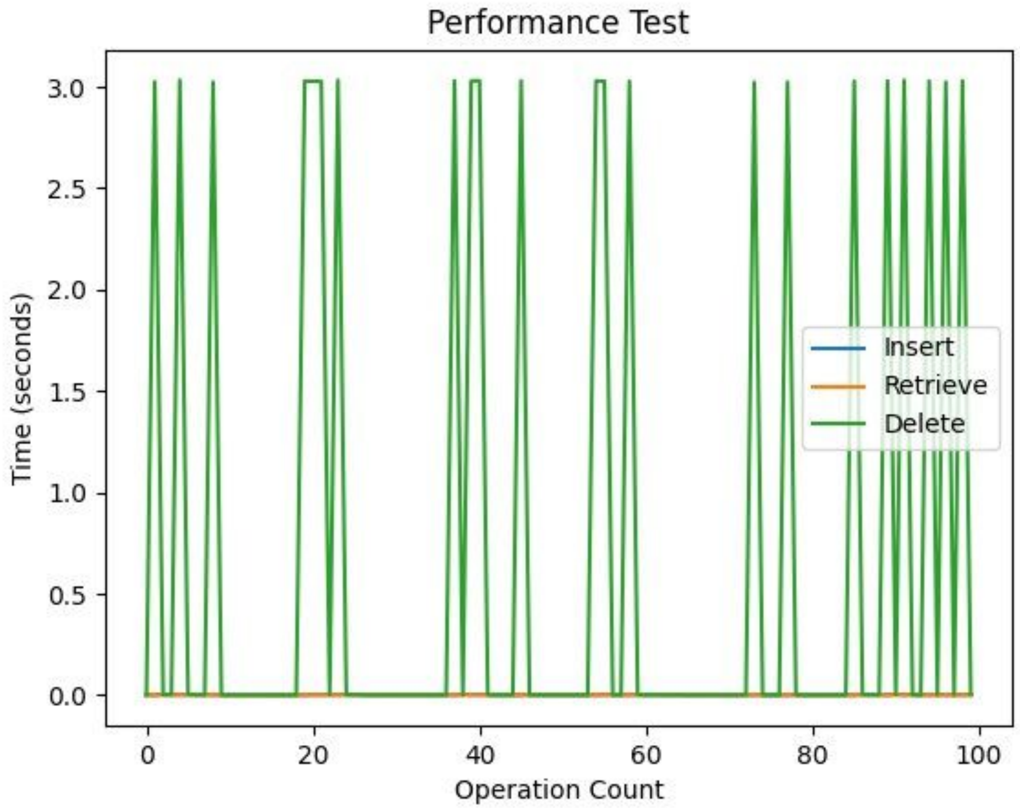
4. Error Handling

- It handle common issues like server unavailability or invalid operations.

5. Node Multithreading

- The Node uses multithreading to handle multiple request simultaneously, ensuring that requests don’t have to wait for the previous request to finish .
- It allows the system to handle multiple requests in parallel and improves performance.

Evaluation


Performance Test
Time (seconds)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
Operation Count
Insert
Retrieve
Delete

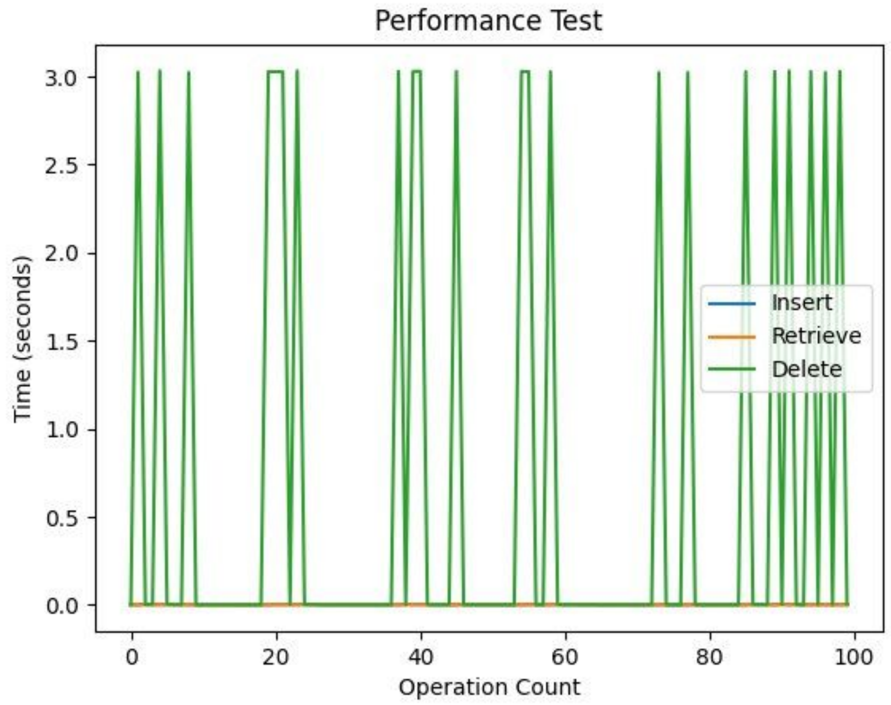
Evaluation

3. Parameters/Configuration (Independent Variables)

- **Insertion time**
- **Deletion time**
- **Retrieval time**

The tesfile calculates the three time parameters Insertion, Deletion and retrieval time and then finally plots the graph of each requests

Time Analysis



Summary of results

In this Analysis we observe that fetching time of key is less than the time of storing and time of deleting key value.

The retrieval requests is faster than the delete operations and the primary reason is that deletion requires the file input output to perform the deletion which is an overhead in the deletion but not in case of retrieval

Unfinished scope

Data Replication :- The data replication to increase fault tolerance.

Multiple Node Failure:- The system fails when multiple node fails at the same time. There is a probability that all the node having the replication might crash and loosing the data.

Updation\Deletion Performance Improvement:- The updation and deletion of keys in system requires to calculate all possible node to ensure consistency which can be improved by Chords algorithm.

Load Distribution:- The load between the nodes might not get well and perfectly distributed over all possible key space.

Challenges

1. Failure Handling

- One of the toughest parts of our project was dealing with temporary server failures. Sometimes, when we try to replicate data,then our server might be down or unavailable. To handle these situation without data loss or inconsistency was challenging.
- We had to design the system which handle failures smoothly by retrying failed operations and ensuring data replication when the server comes back online.

Reflection

What was interesting about the project?

The use of the `scan` module to dynamically discover and manage active nodes in the network adds a practical and often overlooked layer of functionality. This ensures that the system adapts to changes in the cluster without manual intervention

What is one thing that you would have done differently if you had the opportunity to redo this project?

I would like to implement Chord Algorithm in circular DHT with consistent hashing to perform traversal operation in O(logn) time complexity and efficient load distribution.

Conclusions

- Our project is based on distributed key-value store that shows the potential of building a scalable, fault-tolerant, and decentralized data management system. By integrating concepts from distributed systems, networking, and data management, it effectively addresses the key challenges such as data availability, consistency, and fault tolerance in real-world scenarios.

- This project successfully create a lightweight and efficient key-value store that strikes a balance between simplicity and functionality. Although there is room for improvement, it provides a solid base for exploring advanced distributed systems and their real-world use in modern computing.

References

https://www.cs.princeton.edu/courses/archive/spr05/cos598E/bib/dabek-chord.pdf