Final Report: Distributed Database System using Go

Objective

The goal of this project is to design and implement a basic distributed database system using the Go programming language. This system aims to introduce and apply core concepts in distributed systems such as replication, fault tolerance, and communication between distributed nodes.

System Architecture

The architecture consists of at least three nodes: one Master Node and two or more Slave Nodes.

Master Node:

- Has full write access to the database.
- Can create databases and tables dynamically, each with customizable attributes.
- Can perform full CRUD operations.
- Has exclusive access to drop the entire database.
- Broadcasts updates to slave nodes for replication.

Slave Nodes:

- Each slave node contains a read-only copy of the database.
- Capable of executing search, select, update, delete, and insert operations (except dropping the DB).
- Listens to a Message Queue (MQ) or communication stream from the master for data replication.

Communication:

- Nodes communicate over a network using TCP or HTTP protocols.
- The system ensures consistent data propagation from the master node to the slaves.

Final Report: Distributed Database System using Go

Core Features

- Dynamic Database and Table Creation: The master node can create new databases or use existing ones. It can dynamically define tables with specific attributes.
- Query Support Across All Nodes: All nodes (master and slaves) can perform operations such as SELECT, INSERT, UPDATE, DELETE, and table-level operations. Only the master is allowed to drop a database.
- Data Replication: Once data is written or updated in the master, changes are broadcasted to all slave nodes to keep them in sync.

Bonus Features

- Fault Tolerance (Optional): If the master node fails, a slave node can be promoted to become the new master, ensuring continuity.
- Graphical User Interface (Optional): A GUI interface can be implemented using Go libraries like Fyne or cross-language libraries like Flutter or Tkinter (Python). This allows visual management of nodes and databases.

Design Choices

- Go (Golang) was selected for its simplicity, concurrency support, and robust networking capabilities.
- Using TCP/HTTP provides flexibility in network communication.
- The master-slave pattern offers a clear separation of responsibilities, making replication straightforward.
- Emphasis was placed on making the system modular so it can be extended later with load balancing or advanced fault detection.

Challenges Faced

- Synchronizing data across all nodes while avoiding data inconsistency.

- Handling node failures without losing data integrity.

- Ensuring queries on slaves reflect the most recent state from the master.

- Designing a dynamic and flexible way to define table structures at runtime.

- Managing concurrent requests and broadcasting updates efficiently.

Conclusion

This project demonstrates the foundation of a distributed database system with real-world concepts such as replication, fault tolerance, and dynamic schema handling. Using Go's powerful networking and concurrency capabilities, we built a system that simulates how real distributed databases function. The architecture also supports scalability and further improvements such as consensus protocols or distributed transactions.