

Bachelor of Science in Software Engineering

Preventing Data Loss using Raft Consensus Algorithm in a Decentralized Database System

Md. Muhtaseen Hafiz - 180042102

A.K.M Nafiz Zaman - 180042115

Md Shadman Shafi - 180042135

Department of Computer Science and Engineering (CSE)

Islamic University of Technology (IUT)

May, 2023

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Md. Muhtaseen Hafiz, A.K.M Nafiz Zaman and Md. Shadman Shafi under the supervision of Faisal Hussain, Assistant Professor of the Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Md. Muhtaseen Hafiz
Student ID - 180042102

A.K.M Nafiz Zaman
Student ID - 180042115

Md. Shadman Shafi
Student ID - 180042135

Supervisor:

Faisal Hussain
Assistant Professor
Department of Computer Science and Engineering
Islamic University of Technology (IUT)

Acknowledgement

We would like to express our grateful appreciation for **Faisal Hussain**, Assistant Professor, Department of Computer Science & Engineering, IUT for being our adviser and mentor. His motivation, suggestions and insights for this research have been invaluable. Without his support and proper guidance this research would never have been possible. His valuable opinion, time and input provided throughout the thesis work, from first phase of thesis topics introduction, subject selection, proposing algorithm, modification till the project implementation and finalization which helped us to do our thesis work in proper way. We are really grateful to him.

Abstract

In today's digital era, decentralized database management systems have gained significant attention due to their ability to provide scalability, fault tolerance, and improved performance. However, ensuring data integrity, preventing data loss, and maintaining data consistency in such systems remain challenging tasks. This thesis addresses these challenges by proposing a peer-to-peer gossip-based solution that leverages the Raft consensus algorithm and replicated log method.

The proposed solution focuses on making each node in the database cluster a witness to transactions, allowing for consensus on the current state of the database. By utilizing gossip-based protocols, transaction information is disseminated among nodes, ensuring that updates reach all relevant participants. The Raft consensus algorithm is employed to achieve agreement on the committed transactions, while the replicated log method synchronizes transaction logs across all nodes.

The objectives of this thesis include preventing data loss, maintaining data consistency, and meeting high transaction and view request targets. With a target transaction rate of 1000 transactions per second and a target view request rate of 10000 requests per second, the solution aims to deliver robust performance and reliability. By combining the peer-to-peer gossip-based approach, Raft consensus algorithm, and replicated log method, the proposed solution offers benefits such as fault tolerance, scalability, and data consistency.

The thesis contributes to the field by addressing the limitations of current database systems and proposing an innovative solution that ensures data integrity in decentralized environments. The limitations and complexities of Direct Mail, Anti-Entropy, and Rumor Mongering techniques are analyzed, leading to the development of a more effective and efficient solution. The solution's architecture, mechanisms, and protocols are designed to meet the specified targets and provide a reliable foundation for decentralized database management systems.

Through simulations and performance evaluations, the proposed solution demonstrates its effectiveness in preventing data loss, maintaining data consistency, and meeting the specified transaction and view request targets. The results highlight the solution's scalability, fault tolerance, and ability to handle high transaction rates.

In conclusion, this thesis presents a peer-to-peer gossip-based solution that leverages the Raft consensus algorithm and replicated log method to prevent data loss and ensure data consistency in decentralized database management systems. The solution offers a robust and scalable approach, addressing the limitations of existing techniques. With its potential applications in various domains, the proposed solution contributes to the advancement of decentralized database management systems, providing a foundation for reliable and high-performance data storage and processing.

Contents

1	Introduction	5
1.1	Overview	5
1.2	Problem Statement	6
1.3	Motivation	7
1.4	Thesis Objective	8
1.5	Thesis Contributions	9
1.6	Organization of the Thesis	11
2	Literature Review	13
2.1	Chain Replication for Supporting High Throughput and Availability [3]	13
2.1.1	Limitations	14
2.2	A peer-to-peer consensus algorithm to enable storage reliability for a decentralized distributed database [7]	16
2.2.1	Summary	16
3	Background	17
3.1	Data	17
3.2	Data Loss	19
3.3	Data Storage Methods	21
3.3.1	Relational databases	21
3.3.2	NoSQL Databases	22
3.3.3	Object-Oriented Databases	22
3.3.4	In-Memory Databases	22
3.3.5	Columnar Databases	22
3.3.6	Cloud-based Databases	23
3.3.7	NewSQL Databases	23
3.3.8	Decentralized Database Systems	23
3.4	Main Problems of current database systems	24

3.4.1	Scalability	25
3.4.2	Performance Bottlenecks	25
3.4.3	Data Integration	25
3.4.4	Data Security and Privacy	25
3.4.5	High Availability and Fault Tolerance	26
3.4.6	Real-time Analytics	26
3.4.7	Data Consistency	26
3.4.8	Concurrency Control	27
3.4.9	Single Point of Failure	27
3.4.10	Data Security	27
3.4.11	Privacy Protection	28
3.5	Ways of Data loss	28
3.5.1	Hardware or System Failures	28
3.5.2	Software or Application Failures	29
3.5.3	Human Errors	29
3.5.4	Malicious Attacks	29
3.5.5	Natural Disasters	30
3.5.6	Data Corruption	30
3.6	Data Loss prevention strategies	30
3.6.1	Regular Data Backup	31
3.6.2	Data Replication	31
3.6.3	Redundant Storage Systems	31
3.6.4	Disaster Recovery Planning	32
3.6.5	Data Validation and Integrity Checks	32
3.6.6	Security Measures	32
3.6.7	User Education and Training	32
3.7	Database Backup	33
3.7.1	Importance of Database Backup	33
3.7.2	Backup Strategies	34
3.7.3	Considerations for Database Backup	35

3.7.4	Limitations	35
3.8	Database Replication	37
3.8.1	Overview of Database Replication	38
3.8.2	Benefits of Database Replication	38
3.8.3	Approaches to Database Replication	39
3.8.4	Considerations for Database Replication	40
3.8.5	Limitations	41
3.9	Strategies to implement	43
3.9.1	Direct Mail	43
3.9.2	Limitations	44
3.9.3	Complexities	44
3.9.4	Anti-Entropy	45
3.9.5	Limitations	45
3.9.6	Complexities	46
3.9.7	Rumor Mongering	46
3.9.8	Limitations	46
3.9.9	Complexities	47
4	Proposed Structure	48
4.0.1	Replicated LogChain Method	48
4.0.2	Client Authenticity	48
4.0.3	Reaching Consensus with Raft Algorithm Instead of Ac- knowledgement of Database Commit	49
4.1	Benefits of our solution	49
4.2	Proposed Architecture	51
4.3	Client/Maintainer	52
4.4	Entrypoint RPC	52
4.5	Verifier/Node(Cluster)	53
4.6	Execution Scenarios	53
5	Simulation	55

5.1	Implementation Architecture	55
5.2	simulation results	55
6	Result Analysis	56
7	Future Works	58
8	Conclusion	59

1 Introduction

Data loss refers to the irreversible disappearance or corruption of data, leading to the unavailability of crucial information and potential disruptions to business operations.[1] In decentralized database management systems, where data is distributed across multiple nodes, the risk of data loss is amplified due to the increased complexity and reliance on network communications. When a node fails or is compromised, there is a potential for losing the data stored on that node, jeopardizing the system's overall integrity. In today's digital age, databases play a pivotal role in storing and managing vast amounts of critical information. Sensitive data loss can have a negative impact on an organization's long-term stability as well as cause severe reputational and financial damage [2]. However, ensuring data integrity and availability is a persistent challenge, particularly in decentralized database management systems where multiple nodes collaborate to store and process data. Data loss, caused by node failures, network disruptions, or malicious attacks, poses a significant risk to the reliability and trustworthiness of such systems. Therefore, it becomes crucial to develop robust mechanisms that prevent data loss and maintain data consistency in decentralized environments.

1.1 Overview

Data loss is a critical concern in today's digital world, where information is the backbone of businesses and organizations.[2] The need for reliable and secure data management systems has never been more crucial. The introduction provides an overview of the challenges associated with data loss in decentralized databases and emphasizes the importance of developing robust mechanisms to prevent such loss. It highlights the role of databases in storing and managing vast amounts of crucial information and discusses the increased risk of data loss in decentralized environments due to the complexity and reliance on network communications. The primary objective of this research is to prevent data loss and maintain data consistency in decentralized database management systems. The primary objective of

this research is to prevent data loss and maintain data consistency in decentralized database management systems.

1.2 Problem Statement

In decentralized database management systems, ensuring data integrity and availability is a persistent challenge due to the distributed nature of data storage and processing. The risk of data loss is amplified in these systems, where data is distributed across multiple nodes and relies on network communications. When a node fails or is compromised, the data stored on that node becomes inaccessible, potentially leading to data loss and disruptions in business operations. Therefore, there is a crucial need to develop robust mechanisms that prevent data loss and maintain data consistency in decentralized environments.

The existing solutions for data loss prevention and data consistency in centralized database management systems do not directly translate to decentralized systems. Decentralized environments introduce complexities such as node failures, network partitions, and malicious attacks, making it challenging to ensure the reliability and trustworthiness of the data stored and processed across multiple nodes.

The challenge lies in designing and implementing a decentralized database management system that can prevent data loss and maintain data consistency in the face of node failures, network disruptions, and malicious actions. Such a system must provide mechanisms to replicate data across nodes, ensure consensus on the order and validity of transactions, and recover from failures while maintaining high performance and availability.

Additionally, achieving high transaction rates and accommodating a large number of view requests further adds to the complexity. The system needs to handle a high volume of transactions while ensuring timely access to data for various read operations, such as queries and views, without sacrificing data integrity or availability.

Therefore, the problem statement is to develop a decentralized database management system that effectively prevents data loss, maintains data consistency, and supports high transaction rates and view request volumes. The proposed system should leverage appropriate replication and consensus mechanisms to ensure fault tolerance, data redundancy, and timely access to data, while effectively addressing the challenges specific to decentralized environments.

Solving this problem will contribute to the development of robust and reliable decentralized database management systems, enabling businesses and organizations to securely store and process data across multiple nodes while maintaining data integrity, availability, and performance.

1.3 Motivation

The motivation behind this thesis stems from the increasing reliance on decentralized database management systems and the critical need to prevent data loss while maintaining data consistency. In today's digital landscape, businesses and organizations handle vast amounts of data that are crucial for their operations, decision-making processes, and customer interactions. Any loss or corruption of this data can have severe consequences, including financial losses, reputational damage, and disruptions to business continuity.

Decentralized database management systems offer numerous advantages, such as scalability, fault tolerance, and improved performance, by distributing data across multiple nodes. However, these systems also introduce unique challenges in terms of data consistency, data loss prevention, and synchronization among distributed replicas. Existing approaches to data loss prevention and data consistency in decentralized systems have their limitations and may not be able to meet the increasing demands of modern applications and workloads.

Hence, the primary motivation of this thesis is to propose a robust solution that addresses the challenges of data loss prevention and data consistency in decentralized database management systems. By leveraging the peer-to-peer gossip-based

approach, Raft consensus algorithm, and replicated log method, we aim to develop a solution that ensures data integrity, minimizes the risk of data loss, and guarantees consistent views of the database across all nodes.

Moreover, the motivation also arises from the need for high-performance systems that can handle significant transaction rates and view requests. The specified targets of 1000 transactions per second and 10000 view requests per second reflect the requirements of modern applications that operate in dynamic and demanding environments. Our solution aims to meet these targets while ensuring the reliability and consistency of data.

By addressing the limitations and complexities of current database systems, this thesis seeks to contribute to the advancement of decentralized database management systems and provide practical solutions for data loss prevention and data consistency. The proposed solution has the potential to benefit various domains, including finance, healthcare, e-commerce, and distributed applications, where data integrity and availability are of utmost importance.

In conclusion, the motivation behind this thesis is driven by the critical need to prevent data loss, ensure data consistency, and meet the performance demands of decentralized database management systems. By proposing an innovative solution that combines the peer-to-peer gossip-based approach, Raft consensus algorithm, and replicated log method, we aim to make significant contributions to the field and address the challenges faced by modern decentralized systems.

1.4 Thesis Objective

The objective of this thesis is to develop and evaluate a robust solution for preventing data loss and maintaining data consistency in a decentralized database management system. The proposed solution aims to address the challenges specific to decentralized environments, including node failures, network disruptions, and malicious attacks while ensuring high transaction rates and accommodating a large number of view requests.

The primary objectives of this research are as follows:

1. Design a decentralized database management system that leverages a peer-to-peer gossip protocol for information dissemination and the Raft Consensus Algorithm on the Replicated Log Method for achieving consensus on the current state of the database. This design will provide a resilient and efficient approach to data replication, fault tolerance, and recovery.
2. Implement the proposed solution and integrate it into a decentralized database management system prototype. The implementation should consider key factors such as scalability, performance, and fault tolerance to ensure that the system can handle high transaction rates and view request volumes.
3. Evaluate the performance and effectiveness of the implemented solution through comprehensive testing and experimentation. The evaluation should consider factors such as transaction throughput, latency, fault tolerance, and scalability. The results will provide insights into the feasibility and efficiency of the proposed approach in preventing data loss and maintaining data consistency in a decentralized environment.
4. Compare the performance and effectiveness of the proposed solution with existing approaches and highlight the advantages and limitations of the proposed solution. This comparative analysis will contribute to the understanding of the proposed solution's potential impact and its positioning in the context of existing research and practical applications.

1.5 Thesis Contributions

This thesis makes the following contributions to the field of decentralized database management systems:

1. Proposed Solution: This thesis presents a novel solution for preventing data loss and maintaining data consistency in decentralized environments. The proposed solution leverages a peer-to-peer gossip-based protocol for information dissemination and the Raft Consensus Algorithm on the Replicated

Log Method for achieving consensus on the current state of the database. This design offers a robust and efficient approach to data replication, fault tolerance, and recovery in a decentralized context.

2. **System Implementation:** The thesis includes the implementation of the proposed solution as a prototype of a decentralized database management system. The implementation considers crucial factors such as scalability, performance, and fault tolerance to ensure the system can handle high transaction rates and accommodate a large number of view requests. The implemented system serves as a tangible demonstration of the proposed solution's feasibility and functionality.
3. **Performance Evaluation:** Extensive testing and experimentation are conducted to evaluate the performance and effectiveness of the implemented solution. The evaluation considers metrics such as transaction throughput, latency, fault tolerance, and scalability. The results provide insights into the system's efficiency in preventing data loss, maintaining data consistency, and handling high transaction rates and view request volumes in a decentralized environment.
4. **Comparative Analysis:** The thesis includes a comparative analysis of the proposed solution with existing approaches for preventing data loss and maintaining data consistency in decentralized database management systems. This analysis highlights the advantages and limitations of the proposed solution and provides a comprehensive understanding of its potential impact and practical significance. It offers valuable insights for researchers, practitioners, and system designers in selecting appropriate mechanisms for data loss prevention in decentralized environments.
5. **Practical Implications:** The findings of this research have practical implications for various domains that rely on decentralized databases, such as finance, healthcare, and e-commerce. The proposed solution contributes to the development of robust and reliable decentralized database management

systems, ensuring the integrity and availability of crucial information. The research outcomes can guide the design and implementation of decentralized databases, providing organizations with a foundation for secure and efficient data management.

By making these contributions, this thesis advances the knowledge and understanding of data loss prevention and data consistency maintenance in decentralized database management systems. It provides valuable insights and practical solutions to address the challenges specific to decentralized environments, ultimately enhancing the reliability, security, and performance of decentralized databases in real-world applications.

1.6 Organization of the Thesis

The organization of the thesis is as follows:

Chapter 1 of the thesis provides an overview of the thesis, including the problem statement, the motivation behind the research, the objective of the thesis, and the contributions it aims to make. The section concludes with an outline of the organization of the thesis.

In chapter 2, the existing literature and studies related to the topic of the thesis are discussed. It provides a comprehensive review of the relevant research conducted by other scholars in the field.

Chapter 3: Background, presents the background information necessary for understanding the thesis. It covers data, data loss, and various data storage methods, including relational databases, NoSQL databases, object-oriented databases, in-memory databases, columnar databases, cloud-based databases, NewSQL databases, and decentralized database systems. Additionally, it discusses the main problems encountered in current database systems, such as scalability, performance bottlenecks, data integration, data security and privacy, high availability and fault tolerance, real-time analytics, data consistency, concurrency control, and more.

Chapter 4: Proposed Solution, in this chapter, the proposed solution to address the problems identified in Chapter 3 is presented. It discusses the use of a peer-to-peer gossip-based approach, witnessing transactions, reaching consensus with the Raft algorithm, and utilizing the replicated log method. The chapter also highlights the benefits of the proposed solution.

Chapter 5: Conclusion, the final chapter summarizes the main findings of the thesis and provides a conclusion. It discusses the contributions made by the research and suggests potential areas for future exploration or improvement.

By following this organization, the thesis effectively introduces the research problem, explores relevant studies, provides the necessary background information, presents the proposed solution, and concludes the findings, ensuring a logical and coherent structure throughout the document.

2 Literature Review

Database replication is a critical technique used in distributed database systems to ensure data availability, improve performance, and provide fault tolerance. It involves the process of synchronizing or copying data from a source database to one or more target databases. This replication process allows for the creation of redundant copies of data, which can be distributed across multiple servers or locations. Any modifications made to the source database, such as inserts, updates, or deletes, are captured and propagated to the target databases in a controlled and consistent manner. This ensures that the data remains consistent and up-to-date across all replicas. Replication can be implemented using various strategies, such as master-slave replication, multi-master replication, or peer-to-peer replication, each with its own advantages and considerations. Database replication plays a vital role in achieving high availability, scalability, and resilience in modern database systems, enabling organizations to meet the demands of their applications and handle large volumes of data effectively.

2.1 Chain Replication for Supporting High Throughput and Availability [3]

Chain replication is an innovative method used to manage groups of fail-stop storage servers. Its purpose is to facilitate the operation of extensive storage services that require both high throughput and availability while maintaining strong consistency guarantees. In addition to describing the protocols used in chain replication, the study includes simulation experiments to analyze the performance features of a preliminary implementation. The experiments investigate aspects such as throughput, availability, and various object placement strategies, including those based on distributed hash table routing. Windows Azure Store uses this architecture to replicate a database among its services[4].

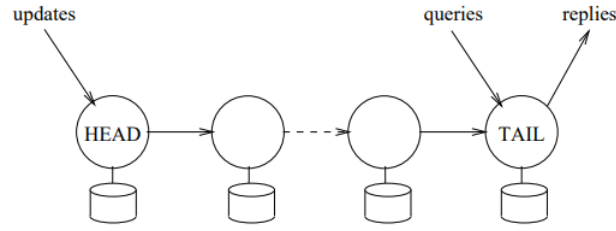


Figure 1: A chain in a replicated chain based database replication system

Reply Generation

The response to each request is created and transmitted by the last element in the chain.

Query Processing

Every query request is directed to the last element in the chain and processed there as a single unit, utilizing the replica of objID stored at that location.

Update Processing

Each update request is sent to the first element in the chain. It is processed there as a single unit using the replica of objID located at the head. The changes in state are then forwarded through a dependable FIFO link to the next element in the chain, where it is processed and forwarded again. This process continues until the request is handled by the last element in the chain.

2.1.1 Limitations

Although this method ensures consistency among all the databases, there is a big delay in the request being propagated to all the servers according to their calculations.

There are multiple studies to efficiently maintain a primary backup for data availability [5] [6] which focuses on keeping a backup of the primary database which also faces the issue of getting acknowledgement from all the available databases.

$\lambda_d = 50$ ms in the head

$\lambda_d = 20$ ms in other servers

$\lambda = 1$ ms in the head

where,

$\lambda_d =$ Committing a query to the database

$\lambda =$ network propagation delay

In total, around 110 ms for one query update for making one transaction which is very slow. The acknowledgement of the commit from all the servers to process the transaction is somewhat very inefficient.

2.2 A peer-to-peer consensus algorithm to enable storage reliability for a decentralized distributed database [7]

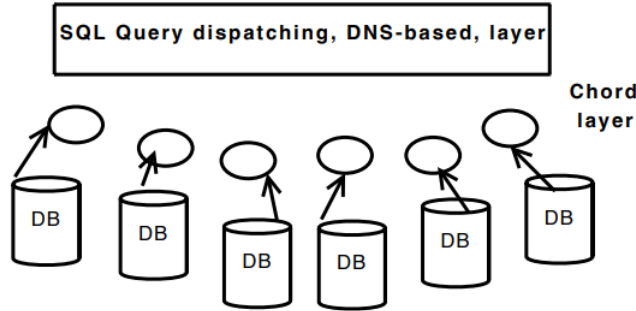


Figure 2: Layered architecture of the distributed database

This research paper addressed the crucial issue of trust within a particular type of distributed system, which focused on a fully decentralized distributed database. The primary objective was to enhance the dependability of data storage by promptly identifying and excluding any compromised or malicious nodes.

To accomplish this, they proposed an innovative approach to a distributed consensus algorithm that is tailor-made for the Chord DHT. The Chord DHT served as the fundamental framework for peer-to-peer communication within the decentralized database server.

Notably, their solution for eliminating malicious nodes demonstrates exceptional efficiency, requiring only a minimal exchange of additional messages. Furthermore, their approach can be seamlessly applied to decentralized distributed systems based on DHTs, broadening its potential utilization beyond the confines of our specific database scenario. figure.

2.2.1 Summary

The paper presents a flexible decentralized distributed database that aims to ensure reliable storage. This database relies on the Chord DHT as a logical infras-

structure, facilitating data location and persistence through replication. The decentralized database comprises three main components: the query dispatching layer, based on DNS static load balancing; the Chord layer, which separates the client from storage nodes; and the storage nodes themselves, running non-distributed versions of database servers.

The database adopts a simple data consistency model, propagating changes to each replica upon data modification. It supports INSERTs, SELECTs, and DELETEs but lacks UPDATE functionality. The absence of UPDATEs simplifies replica consistency, with DELETEs requiring the removal of all corresponding replicas.

While the decentralized database demonstrates better handling of node volatility and data persistence compared to solutions like MySQL Cluster, reliability concerns arise due to potentially malicious Chord nodes capable of concealing or altering data. Traditional methods like authentication with a centralized server undermine the advantages of a fully decentralized storage system. Alternative methods exist for identifying malicious nodes within decentralized systems, although they may be more complex to implement. These methods leverage collaboration between nodes and are rooted in game theory and distributed consensus.

3 Background

In this section, we provide a comprehensive overview of the relevant studies and existing literature that form the foundation for our research on data loss prevention. These studies contribute valuable insights and knowledge in the field, addressing various aspects related to data loss and prevention strategies. By reviewing these studies, we gain a deeper understanding of the challenges and existing solutions, which helps to contextualize and inform our own research.

3.1 Data

Data plays a pivotal role in today’s digital age, serving as the foundation for decision-making, analysis, and operations across various domains [8]. In the con-

text of database management systems, data represents the structured and organized information that is stored, processed, and retrieved by applications and users. It encompasses a wide range of entities, such as records, documents, multimedia files, and more, that collectively form the informational backbone of organizations and systems.

Data holds immense value as it represents the knowledge, insights, and experiences accumulated by individuals, businesses, and societies.[9] It serves as the basis for critical operations, including financial transactions, inventory management, customer relationship management, and scientific research. As such, the integrity, availability, and confidentiality of data are of utmost importance to ensure the smooth functioning and success of organizations.

Integrity refers to the accuracy, consistency, and trustworthiness of data. It ensures that the information stored in the database is reliable and free from errors or unauthorized modifications. Data integrity is crucial to maintain the validity and reliability of business processes, analytical reports, and decision-making.[10] Any loss or corruption of data can have severe consequences, leading to incorrect insights, disrupted operations, and compromised trust.

Availability refers to the accessibility of data when needed. It implies that authorized users and applications can retrieve and interact with the data in a timely and uninterrupted manner. High availability is essential to support real-time operations, online transactions, and responsive user experiences. Data unavailability can result in service disruptions, financial losses, and negative impacts on customer satisfaction and trust.

Confidentiality relates to the protection of sensitive or private data from unauthorized access or disclosure. In many domains, such as healthcare, finance, and personal information management, data confidentiality is crucial to comply with legal and ethical requirements. Breaches in data confidentiality can lead to privacy violations, identity theft, reputational damage, and legal repercussions.[11]

In decentralized database management systems, the challenges related to data integrity, availability, and confidentiality are magnified due to the distributed nature of data storage and processing. The reliance on network communications and the presence of multiple nodes introduce complexities and vulnerabilities. The risk of data loss increases with node failures, network disruptions, and potentially malicious actions.[12]

Addressing these challenges requires the development of robust mechanisms for data replication, fault tolerance, and recovery. Solutions must ensure that data remains consistent across multiple nodes, even in the presence of failures or inconsistencies in communication. Moreover, mechanisms must enable efficient and secure access to data, accommodating high transaction rates and a large number of view requests while maintaining data integrity and availability.

By addressing the complexities and challenges associated with data in decentralized environments, this thesis aims to contribute to the development of reliable and efficient decentralized database management systems. The proposed solution focuses on preventing data loss, maintaining data consistency, and enhancing the overall reliability and performance of decentralized databases, ultimately ensuring the integrity and availability of crucial information for organizations and users.

3.2 Data Loss

Data loss refers to the irreversible disappearance or corruption of data, leading to the unavailability and potential destruction of critical information. It is a significant concern for organizations and individuals alike, as data loss can have severe consequences, including financial losses, operational disruptions, reputational damage, and legal ramifications.

In the context of decentralized database management systems, data loss poses an even greater risk due to the distributed nature of data storage and processing. In these systems, data is distributed across multiple nodes, and each node holds a portion of the overall data set. This distribution is designed to enhance scalability,

fault tolerance, and performance. However, it also introduces vulnerabilities and complexities that can result in data loss.

Several factors contribute to the risk of data loss in decentralized environments. Node failures, whether due to hardware malfunctions, software errors, or power outages, can lead to the loss of data stored on that particular node. Network disruptions, such as connectivity issues or network partitions, can prevent data synchronization and replication across nodes, potentially resulting in data inconsistencies or loss. Additionally, malicious attacks, including hacking, data breaches, or ransomware, can compromise the security and integrity of data, leading to data loss or unauthorized access.

The consequences of data loss can be far-reaching. In business settings, organizations may lose valuable customer information, financial records, intellectual property, or transactional data. This can lead to financial setbacks, customer dissatisfaction, legal complications, and damage to the organization's reputation. In sectors such as healthcare or research, data loss can have life-threatening implications, as critical patient data or scientific findings may be permanently destroyed or become inaccessible.

Preventing data loss in decentralized database management systems requires robust mechanisms and strategies. Data replication is a fundamental approach to mitigate the risk of data loss by creating redundant copies of data across multiple nodes. This redundancy ensures that even if one node fails, the data can still be retrieved from other nodes, minimizing the impact of data loss. Additionally, implementing fault tolerance mechanisms such as backup systems, data recovery procedures, and disaster recovery plans can help mitigate the impact of data loss and expedite the restoration process.

Furthermore, implementing strong security measures, including access controls, encryption, and intrusion detection systems, can protect data from unauthorized access and mitigate the risk of data loss due to malicious attacks. Regular data

backups, both locally and off-site, can provide additional layers of protection, ensuring that data can be restored even in the event of catastrophic failures.

The research presented in this thesis aims to develop a solution that effectively prevents data loss in a decentralized database management system. By leveraging a peer-to-peer gossip-based protocol and the Raft Consensus Algorithm on the Replicated Log Method, the proposed solution aims to ensure data replication, fault tolerance, and recovery mechanisms that mitigate the risk of data loss and maintain data consistency in the face of node failures, network disruptions, and malicious actions. The objective is to provide organizations with a reliable and resilient decentralized database management system that minimizes the likelihood and impact of data loss, ensuring the integrity, availability, and confidentiality of critical information.

3.3 Data Storage Methods

The choice of a database storage system depends on specific requirements such as data volume, access patterns, scalability, consistency, and performance needs. Each type of database storage system has its strengths and trade-offs, and the selection should be based on a thorough analysis of the application's needs and the desired characteristics of the data management solution.[13]

3.3.1 Relational databases

Relational databases are based on the relational model and have been widely used for decades. They store data in tables with predefined schemas, where relationships between tables are defined through keys. Examples of popular RDBMS include MySQL, Oracle Database, and Microsoft SQL Server. RDBMS offer strong data consistency, transactional support, and support for complex querying using SQL.

3.3.2 NoSQL Databases

NoSQL databases are designed to handle large-scale, distributed, and unstructured data. Unlike RDBMS, they do not adhere to a fixed schema and offer flexible data models. NoSQL databases are classified into different types, including document-oriented databases (e.g., MongoDB), key-value stores (e.g., Redis), columnar databases (e.g., Apache Cassandra), and graph databases (e.g., Neo4j). NoSQL databases excel in scalability, high availability, and performance but may sacrifice strong consistency guarantees.

3.3.3 Object-Oriented Databases

Object-oriented databases (OODBMS) store data in the form of objects, similar to object-oriented programming languages. They provide better support for complex data structures and encapsulate both data and behavior. OODBMS offer more natural mapping for object-oriented applications, enabling direct storage and retrieval of objects without the need for complex mapping to relational tables. Examples of OODBMS include db4o and Versant.

3.3.4 In-Memory Databases

In-memory databases (IMDB) store data entirely in the main memory, providing fast data access and query processing. They eliminate disk I/O bottlenecks and are well-suited for applications requiring low-latency, high-throughput operations. IMDBs often employ techniques like data compression and caching to optimize memory utilization. Examples of in-memory databases include Redis, Memcached, and Apache Ignite.

3.3.5 Columnar Databases

Columnar databases store data in a column-oriented fashion rather than the traditional row-based storage of RDBMS. They offer improved query performance for analytical workloads, as they only read the necessary columns for a query, reducing I/O overhead. Columnar databases excel in handling large datasets and

performing complex analytical queries efficiently. Popular columnar databases include Apache Cassandra, Apache HBase, and Google Bigtable.

3.3.6 Cloud-based Databases

Cloud-based databases are hosted on cloud platforms and provide scalable and on-demand database services. They offer the benefits of automatic scalability, high availability, and reduced administrative overhead. Examples include Amazon Web Services (AWS) DynamoDB, Google Cloud Spanner, and Microsoft Azure Cosmos DB. Cloud-based databases are well-suited for cloud-native applications and can seamlessly integrate with other cloud services.

3.3.7 NewSQL Databases

NewSQL databases aim to bridge the gap between traditional RDBMS and NoSQL databases. They provide the scalability and fault tolerance of NoSQL databases while maintaining ACID (Atomicity, Consistency, Isolation, Durability) properties. NewSQL databases optimize distributed query processing and offer horizontal scalability without compromising on data consistency. Examples include CockroachDB, NuoDB, and TiDB.

3.3.8 Decentralized Database Systems

Decentralized database systems are designed to distribute data across multiple nodes in a network, eliminating the need for a central authority or single point of failure.[14] Each node in a decentralized database system stores a portion of the data, and coordination mechanisms are employed to ensure data consistency and availability. These systems leverage peer-to-peer (P2P) architectures and distributed consensus algorithms to achieve fault tolerance and scalability.

Blockchain-based databases, such as Bitcoin and Ethereum, are prominent examples of decentralized databases. They utilize distributed ledgers and consensus mechanisms to maintain a decentralized and tamper-resistant record of transac-

tions. Blockchain databases are characterized by their immutability, transparency, and resilience against malicious attacks.

Other decentralized database systems, such as IPFS (InterPlanetary File System) and BigchainDB, provide decentralized storage and retrieval of data while offering features like content addressing, data versioning, and distributed querying. These systems aim to address the challenges of data integrity, availability, and privacy in decentralized environments, enabling peer-to-peer data sharing and collaboration without relying on a central authority.

Decentralized database systems offer benefits such as increased data resilience, censorship resistance, and improved fault tolerance. They are particularly suitable for applications that require trustless interactions, data sovereignty, and distributed governance. However, they also present challenges related to consensus protocols, data synchronization, and performance overhead due to the decentralized nature of data storage and processing.

The research presented in this thesis focuses on addressing the challenges specific to decentralized database management systems. By leveraging a peer-to-peer gossip-based protocol and the Raft Consensus Algorithm on the Replicated Log Method, the proposed solution aims to provide a robust and efficient approach to data replication, fault tolerance, and recovery in decentralized environments. The objective is to ensure data consistency, prevent data loss, and enhance the overall reliability and performance of decentralized databases.

3.4 Main Problems of current database systems

Current database systems, while highly effective and widely used, still face several challenges that impact their performance, scalability, and ability to meet evolving data management needs. This section highlights some of the main problems encountered in current database systems.

3.4.1 Scalability

Scalability refers to a database system's ability to handle increasing data volumes, user loads, and transaction rates without compromising performance.[15] Traditional database systems often struggle to scale horizontally due to their reliance on centralized architectures and rigid schemas. Scaling up hardware resources can be expensive and may not provide a linear performance improvement. Additionally, ensuring data consistency and maintaining high availability across distributed environments pose significant challenges.

3.4.2 Performance Bottlenecks

Database systems frequently encounter performance bottlenecks, such as slow query execution, inefficient indexing, or disk I/O limitations. As data volumes grow and complex analytical queries are executed, these bottlenecks can severely impact response times and overall system performance. Improving performance often requires advanced optimization techniques, query tuning, and data partitioning strategies, which can be time-consuming and complex to implement.

3.4.3 Data Integration

In today's data-driven landscape, organizations often deal with heterogeneous data sources, including structured, semi-structured, and unstructured data from various systems and applications. Integrating and harmonizing these disparate data sources into a unified view poses significant challenges. Data integration involves handling schema mismatches, data transformation, and maintaining data consistency across different systems. Traditional database systems may struggle to efficiently handle diverse data types and provide seamless integration capabilities.

3.4.4 Data Security and Privacy

Ensuring data security and privacy is a critical concern in database systems. Unauthorized access, data breaches, and insider threats can compromise sensitive information and result in severe financial and reputational damage. Current systems

rely on access control mechanisms, encryption, and auditing to protect data, but emerging threats and evolving compliance requirements continue to present challenges. Moreover, privacy concerns related to personal data and regulatory frameworks like GDPR require more sophisticated approaches to data anonymization and consent management.

3.4.5 High Availability and Fault Tolerance

Database systems must provide high availability and fault tolerance to minimize service disruptions and data loss. Traditional systems may rely on a single point of failure, such as a central server, which poses a risk to system availability. Achieving fault tolerance often requires complex replication strategies, distributed architectures, and efficient mechanisms for data synchronization and consistency across multiple nodes.[16]

3.4.6 Real-time Analytics

With the exponential growth of data, organizations increasingly demand real-time analytics capabilities to gain timely insights and make informed decisions. Traditional database systems, optimized for transactional workloads, may struggle to provide efficient real-time analytics due to the need for complex joins, aggregations, and ad hoc querying. Balancing the requirements of transactional processing and analytical querying poses a challenge in traditional systems.

3.4.7 Data Consistency

Maintaining data consistency is a crucial aspect of database systems. In multi-user environments, concurrent transactions can access and modify the same data simultaneously, leading to potential data inconsistencies. Ensuring atomicity, consistency, isolation, and durability (ACID properties) is a challenge, especially in distributed or parallel database systems. Coordinating concurrent transactions and resolving conflicts to maintain data consistency requires robust concurrency control mechanisms and careful transaction management.

3.4.8 Concurrency Control

Database systems must handle concurrent access to shared data while ensuring data integrity and avoiding conflicts. Concurrency control mechanisms, such as locking, timestamp ordering, or optimistic concurrency control, aim to coordinate concurrent transactions and prevent issues like lost updates, dirty reads, or non-repeatable reads. However, managing concurrency effectively without sacrificing performance or introducing excessive locking overhead remains a challenge, particularly in high-throughput systems with large numbers of concurrent transactions.

3.4.9 Single Point of Failure

Traditional database systems often rely on a single point of failure, such as a central server or primary node, which poses a risk to system availability and data integrity. If the central server fails or experiences a disruption, the entire system may become unavailable, resulting in significant downtime and potential data loss. Achieving high availability and fault tolerance requires the elimination of single points of failure through distributed architectures, replication, and redundancy strategies.

3.4.10 Data Security

Data security is a critical concern for database systems, given the increasing frequency and sophistication of cyber threats. Database systems must protect data from unauthorized access, data breaches, and insider attacks. Common security measures include user authentication, access controls, data encryption, and auditing. However, security challenges persist due to evolving threats, inadequate security practices, and compliance requirements. Protecting against vulnerabilities, ensuring secure data transmission, and implementing robust security measures are ongoing challenges in the database field.

3.4.11 Privacy Protection

Database systems handle vast amounts of sensitive and personal data, necessitating privacy protection mechanisms. Privacy concerns include data anonymization, ensuring compliance with regulations like GDPR, and providing user consent management. Anonymizing data while maintaining data utility and managing consent in a granular and transparent manner are challenges that require innovative privacy-enhancing technologies within database systems.

Addressing these challenges requires continuous research and innovation in database technologies. New approaches, such as distributed and decentralized databases, distributed consensus algorithms, multi-version concurrency control, advanced encryption techniques, and privacy-preserving mechanisms, are being explored to overcome these limitations. By addressing data consistency, concurrency control, eliminating single points of failure, and enhancing security and privacy measures, the next generation of database systems aims to provide robust, reliable, and secure data management solutions for a wide range of applications.

3.5 Ways of Data loss

Data loss can occur in various ways, leading to the irreversible disappearance or corruption of critical information. Understanding the different ways in which data loss can occur is crucial for implementing effective preventive measures. This section explores some common ways data can be lost in database systems.

3.5.1 Hardware or System Failures

Hardware failures, such as disk crashes, power outages, or server malfunctions, can result in data loss. If a database system does not have proper redundancy or backup mechanisms in place, data stored on the failed hardware or system can become inaccessible or permanently lost. Hardware failures can occur unexpectedly, making it essential to have reliable backup strategies and disaster recovery plans to mitigate the impact of such failures.

3.5.2 Software or Application Failures

Software or application failures, including bugs, glitches, or crashes, can lead to data loss. In some cases, software errors or improper handling of transactions can result in data corruption or incomplete updates, causing inconsistencies in the database. Software failures can occur due to programming errors, compatibility issues, or inadequate error handling. Ensuring robust software design, thorough testing, and error recovery mechanisms are important to minimize the risk of data loss.

3.5.3 Human Errors

Human errors are a common cause of data loss. Accidental deletion, overwriting data, or misconfiguration of database settings can lead to significant data loss. Human errors can occur during routine database administration tasks, data entry processes, or maintenance operations. Proper training, access controls, and data validation mechanisms can help reduce the likelihood of human errors causing data loss.

3.5.4 Malicious Attacks

Malicious attacks, such as hacking, ransomware, or insider threats, pose a significant risk to data security and can result in data loss. Attackers may gain unauthorized access to a database system and intentionally delete or modify data, rendering it unusable. Ransomware attacks can encrypt data, making it inaccessible until a ransom is paid. Insider threats, including disgruntled employees or unauthorized access by individuals with privileged access, can also lead to data loss. Implementing robust security measures, such as access controls, encryption, intrusion detection systems, and regular security audits, can help mitigate the risk of data loss due to malicious attacks.

3.5.5 Natural Disasters

Natural disasters, such as fires, floods, earthquakes, or hurricanes, can physically damage data centers or infrastructure, leading to data loss. Catastrophic events can destroy hardware, disrupt power supply, or cause irreparable damage to storage media. Geographic redundancy, off-site backups, and disaster recovery plans that include data replication to remote locations are essential for mitigating the impact of natural disasters on data loss.

3.5.6 Data Corruption

Data corruption can occur due to various factors, including software bugs, hardware malfunctions, or transmission errors. Corruption can result in the loss of data integrity, making it unusable or unreliable. Corrupted data may lead to incorrect results, system crashes, or the inability to access critical information. Regular data backups, data validation checks, and integrity checks can help identify and mitigate data corruption issues.

Understanding the different ways data loss can occur enables organizations to develop comprehensive strategies for preventing data loss and implementing robust data protection mechanisms. By addressing hardware and software failures, mitigating human errors, implementing strong security measures, preparing for natural disasters, and actively monitoring and detecting data corruption, organizations can minimize the risk of data loss and ensure the availability and integrity of their critical data.

3.6 Data Loss prevention strategies

Data loss prevention strategies are crucial for mitigating the risk of data loss and ensuring the availability and integrity of critical information. Here are some commonly employed strategies in the field:

3.6.1 Regular Data Backup

Implementing a regular data backup strategy is essential for preventing data loss. This involves creating duplicate copies of data and storing them in separate locations, ensuring that in the event of data loss, the backup copies can be used for recovery. Backup schedules should be defined based on the organization's recovery point objective (RPO), which determines how frequently backups should be performed to minimize data loss. Organizations may choose to perform full backups periodically or adopt incremental or differential backup methods to optimize storage space and backup time.

3.6.2 Data Replication

Data replication involves creating and maintaining copies of data across multiple storage devices or locations. Replication improves data availability and ensures redundancy, reducing the risk of data loss in the event of hardware failures or disasters. Replication can be achieved through techniques like database mirroring, log shipping, or using distributed databases that replicate data across multiple nodes.[17] Replication can be synchronous (immediate replication) or asynchronous (delayed replication) based on the desired trade-off between data consistency and performance.

3.6.3 Redundant Storage Systems

Redundant storage systems involve deploying multiple storage devices or servers to store data. Redundancy ensures that if one storage device fails, data remains accessible from other devices. Redundancy can be achieved through techniques such as RAID (Redundant Array of Independent Disks), which distributes data across multiple disks for improved fault tolerance and performance. Redundant storage systems enhance data availability and can mitigate the risk of data loss caused by hardware failures.

3.6.4 Disaster Recovery Planning

Disaster recovery planning involves developing comprehensive strategies and procedures to recover data and restore operations in the event of a catastrophic event. This includes creating backup sites or data centers in geographically separate locations, ensuring data replication to remote locations, and defining recovery time objectives (RTO) to establish the maximum acceptable downtime. Disaster recovery plans should encompass data backup and restoration procedures, hardware and software requirements, communication plans, and regular testing and validation to ensure their effectiveness.

3.6.5 Data Validation and Integrity Checks

Implementing data validation mechanisms, such as checksums or hash functions, helps ensure data integrity and detect potential data corruption or tampering. Regular integrity checks can identify inconsistencies or errors in stored data and facilitate early detection and remediation of data issues before they lead to data loss.

3.6.6 Security Measures

Robust security measures play a vital role in data loss prevention. This includes implementing access controls, encryption mechanisms, intrusion detection systems, and security audits to protect data from unauthorized access, malicious attacks, or insider threats. By safeguarding the confidentiality, integrity, and availability of data, security measures help prevent data loss due to security breaches.

3.6.7 User Education and Training

Educating and training users on data handling best practices, security protocols, and the importance of data backup and replication can significantly

contribute to data loss prevention. Users should be made aware of potential risks, encouraged to follow secure data management practices, and trained on proper backup and recovery procedures.

By implementing a combination of these data loss prevention strategies, organizations can significantly reduce the likelihood and impact of data loss incidents. Each strategy should be tailored to the specific requirements, risk profile, and resources of the organization to ensure an effective and comprehensive data loss prevention approach.

3.7 Database Backup

Database backup is a critical component of data management and plays a vital role in ensuring the availability, recoverability, and integrity of data.[18] It involves creating duplicate copies of the database and storing them in separate locations to protect against data loss and facilitate data recovery in case of system failures, human errors, or other unforeseen events. This section discusses the importance of database backup, backup strategies, and considerations for implementing an effective backup solution.

3.7.1 Importance of Database Backup

Database backup is essential for several reasons:

- a. **Data Recovery:** In the event of data loss due to hardware failures, software errors, or human mistakes, database backups serve as a reliable source for restoring lost or corrupted data. Backups allow organizations to recover data to a known point in time, minimizing downtime and data loss.
- b. **Business Continuity:** Database backups contribute to business continuity by providing a means to restore critical systems and resume operations swiftly. By having up-to-date backups, organizations can quickly recover from disruptive events and maintain continuity in service delivery.

- c. **Compliance and Legal Requirements:** Many industries have regulatory or legal obligations regarding data retention and availability. Database backups help meet these requirements by ensuring that data is safely stored and accessible for compliance audits or legal inquiries.
- d. **Disaster Recovery:** Database backups play a crucial role in disaster recovery scenarios, such as natural disasters or cyber-attacks. Off-site backups enable the restoration of data in alternative locations, mitigating the impact of localized incidents and minimizing data loss.

3.7.2 Backup Strategies

Implementing an effective database backup strategy involves considering various factors, including:

- a. **Backup Frequency:** Organizations need to determine the appropriate backup frequency based on their Recovery Point Objective (RPO), which defines acceptable data loss in case of failure. The backup frequency should align with the RPO to minimize data loss. Common approaches include daily backups, hourly backups, or near-real-time backups.
- b. **Full, Incremental, and Differential Backups:** Backup strategies often combine different backup types. Full backups capture the entire database, while incremental backups store only the changes made since the last full or incremental backup. Differential backups capture changes made since the last full backup. Combining these backup types optimizes storage space and reduces backup time.
- c. **Storage Media:** Organizations can choose different storage media for database backups, such as tapes, disks, or cloud storage. Factors to consider include capacity, access speed, cost, and durability. Cloud storage offers scalability, accessibility, and redundancy, making it an increasingly popular choice for backup storage.

- d. **Retention Period:** Determining the retention period for backups is crucial. It should align with organizational requirements, compliance regulations, and data recovery needs. Retaining backups for an appropriate duration ensures the availability of historical data and facilitates long-term data analysis or auditing.
- e. **Testing and Validation:** Regular testing and validation of backups are essential to ensure their reliability and effectiveness. Organizations should periodically restore backups to test the restoration process and validate the integrity and usability of the backup data.

3.7.3 Considerations for Database Backup

When implementing a database backup solution, several considerations should be taken into account:

- a. **Backup Storage Security:** Backup data should be protected with appropriate security measures, including access controls, encryption, and regular vulnerability assessments. Safeguarding backup storage prevents unauthorized access or tampering with critical data.
- b. **Automation and Monitoring:** Implementing automated backup processes reduces the risk of human errors and ensures consistent backup execution. Monitoring backup operations provides insights into backup success rates, alerts for failures, and enables timely remediation.
- c. **Off-Site and Remote Backups:** Storing backups in off-site or remote locations enhances data protection by mitigating the impact of localized disasters or physical damage to the primary site. Off-site backups should be stored.

3.7.4 Limitations

While database backup is a crucial aspect of data management and disaster recovery, it is important to recognize its limitations. Understanding these limitations

helps organizations develop a comprehensive data protection strategy that addresses potential gaps. Here are some limitations of database backup:

1. **Data Loss Window:** Even with frequent backup schedules, there is always a time window between backups where data changes may not be captured. If a failure or data loss occurs during this window, the most recent changes may not be recoverable. Organizations need to carefully consider their Recovery Point Objective (RPO) and backup frequency to minimize the potential data loss within this window.
2. **Backup and Recovery Time:** Database backups and recoveries can be time-consuming processes, particularly for large databases. The time required to create backups and restore data to the primary system can lead to downtime and impact business operations. Organizations need to balance the backup and recovery time with their Recovery Time Objective (RTO) to minimize the impact on business continuity.
3. **Storage Requirements:** Database backups consume storage space, especially when considering multiple backup copies, retention periods, and incremental backups. Storing backups for extended periods or maintaining multiple copies can result in significant storage costs. Organizations need to assess their storage capacity and consider cost-effective solutions such as data deduplication, compression, or cloud-based storage.
4. **Backup Consistency and Integrity:** Ensuring the consistency and integrity of backup data is crucial. Backup processes need to capture the database state accurately and completely to guarantee successful recovery. Inconsistent or corrupted backups can render the recovery process ineffective, resulting in data loss or incomplete restoration.
5. **Single Point of Failure:** In traditional backup approaches, relying on a single backup server or location creates a single point of failure. If the backup server or storage experiences a failure or is compromised, it can lead to the

loss of both the primary data and the backup data. Organizations should consider implementing redundant backup solutions or off-site backups to mitigate the risk of a single point of failure.

6. **Manual Errors and Misconfigurations:** Human errors, such as misconfigurations or accidental deletions, can affect the effectiveness of database backups. Improper backup settings, incorrect storage locations, or accidental overwriting of backups can compromise the backup process. Organizations should implement proper training, standard operating procedures, and regular reviews to minimize the likelihood of human errors in backup procedures.
7. **Compatibility and Versioning:** Compatibility issues can arise when restoring backups to different database versions or platforms. If the backup is not compatible with the restored environment, it may require additional steps or data transformations for successful recovery. Organizations should consider versioning and compatibility concerns when planning backup and recovery processes.
8. **Backup Testing and Validation:** While backups are created with the intention of data recovery, their reliability and effectiveness can only be confirmed through regular testing and validation. Failure to test backups periodically may result in backups that cannot be successfully restored, rendering the entire backup strategy ineffective.

Understanding these limitations allows organizations to proactively address potential challenges and incorporate additional data protection measures. Implementing complementary strategies such as data replication, real-time synchronization, high availability configurations, or continuous data protection can help overcome these limitations and provide a more robust and comprehensive data protection solution.

3.8 Database Replication

Database replication is a fundamental technique in decentralized database management systems that involves creating and maintaining multiple copies of a database

across different nodes or locations. It plays a crucial role in improving data availability, scalability, and fault tolerance.[19] This section provides an overview of database replication, its benefits, approaches, and considerations for implementing an effective replication strategy.

3.8.1 Overview of Database Replication

Database replication is the process of creating and synchronizing multiple copies of a database to ensure data redundancy and availability. Each copy, also known as a replica, contains the same data and is stored on separate nodes or servers within a distributed database system. Replication enables concurrent access to data, enhances fault tolerance, and improves overall system performance.

3.8.2 Benefits of Database Replication

Database replication offers several benefits for decentralized database management systems:

1. **Improved Data Availability:** Replication increases data availability by allowing multiple replicas to serve read requests. If one replica becomes unavailable due to a node failure or network issue, the remaining replicas can continue serving data, ensuring uninterrupted access for users.
2. **Scalability and Load Balancing:** Replication facilitates horizontal scalability by distributing read operations across multiple replicas. This distributes the query load and improves system performance, enabling the system to handle increased user demand without sacrificing responsiveness.
3. **Fault Tolerance and High Availability:** Replication provides fault tolerance by ensuring that data remains accessible even if individual nodes or replicas fail. If a replica becomes unavailable, other replicas can continue serving data, reducing the impact of failures and minimizing downtime.
4. **Geographical Redundancy:** By replicating data across geographically dispersed locations, database replication provides redundancy and disaster re-

covery capabilities. In the event of a natural disaster, network outage, or localized failure, data can be accessed from replicas in alternative locations.

5. **Data Locality and Performance Optimization:** Replication allows data to be stored closer to users or applications, reducing latency and improving response times. By placing replicas strategically, organizations can optimize data access based on geographical proximity or specific user requirements.

3.8.3 Approaches to Database Replication

Different approaches can be used for database replication, depending on the requirements and characteristics of the system. Some common replication approaches include:

1. **Snapshot Replication:** Snapshot replication involves taking periodic snapshots of the entire database and distributing them to replicas. This approach provides a consistent view of the data at a specific point in time but requires significant data transfer and storage resources.
2. **Transactional Replication:** Transactional replication replicates individual database transactions from the source to the replicas in real-time or near-real-time. This approach ensures that replicas stay synchronized with the source database, maintaining data consistency across all replicas.
3. **Merge Replication:** Merge replication allows bidirectional synchronization of data across replicas. It accommodates scenarios where data modifications can occur on multiple replicas and resolves conflicts during synchronization.
4. **Peer-to-Peer Replication:** Peer-to-peer replication enables each node in a decentralized database system to act as both a source and a replica. It allows data modifications at any replica, and changes are propagated to other replicas in a distributed and decentralized manner.

3.8.4 Considerations for Database Replication

Implementing a successful database replication strategy requires careful consideration of the following aspects:

1. **Consistency and Conflict Resolution:** Ensuring data consistency across replicas is crucial. Techniques such as conflict detection and resolution mechanisms, timestamp-based ordering, or consensus algorithms help maintain data integrity and resolve conflicts that may arise due to concurrent updates.
2. **Replication Topology:** Choosing an appropriate replication topology depends on the system requirements, performance considerations, and fault tolerance objectives. Common topologies include master-slave replication, master-master replication, or multi-master replication.
3. **Replication Lag:** Replication introduces a delay between the source database and replicas, known as replication lag. Organizations need to monitor and manage replication lag to ensure that replicas remain up to date and provide timely data access.
4. **Network Bandwidth and Latency:** Database replication involves transferring data between nodes, which requires sufficient network bandwidth and low latency. Organizations should assess network capacity and latency requirements to avoid performance bottlenecks and data synchronization delays.
5. **Replication Monitoring and Maintenance:** Regular monitoring of replication status, performance, and consistency is essential to identify and resolve any issues promptly. Organizations should have monitoring tools and processes in place to ensure the health and effectiveness of the replication system.

Database replication is a powerful technique that enhances data availability, scalability, and fault tolerance in decentralized database management systems. By

implementing an appropriate replication strategy and considering the specific requirements of the system, organizations can improve data accessibility, system performance, and overall data management capabilities.

3.8.5 Limitations

While database replication offers numerous benefits, it also has certain limitations that organizations should consider when implementing a replication strategy. Understanding these limitations helps ensure that replication meets the desired objectives and aligns with the system's requirements. Here are some limitations of database replication:

1. **Replication Complexity:** Implementing and managing database replication can be complex, especially in large-scale distributed environments. Configuring replication topologies, handling conflicts, managing replication lag, and ensuring data consistency require expertise and careful planning. Organizations need skilled administrators and appropriate tools to effectively deploy and maintain replication systems.
2. **Replication Lag:** Replication introduces a delay between the source database and replicas, known as replication lag. This lag can vary based on factors such as network bandwidth, latency, and the volume of data being replicated. In situations where real-time data access is critical, replication lag can be a limitation, as the replicas might not reflect the most up-to-date data.
3. **Consistency and Conflict Resolution:** Maintaining data consistency across replicas can be challenging, especially when updates occur simultaneously on different replicas. Conflict resolution mechanisms are required to handle conflicts that arise when conflicting changes are made on different replicas. Implementing effective conflict resolution strategies can be complex and require careful consideration of the application's requirements and data semantics.

4. **Scalability and Performance Impact:** Database replication can introduce performance overhead, particularly during write operations. Replicating write transactions across multiple replicas requires additional resources and can impact the system's overall performance. Organizations need to carefully design and optimize their replication infrastructure to ensure scalability and minimize the performance impact on the source database and replicas.
5. **Storage and Network Requirements:** Replicating a database across multiple nodes increases storage and network bandwidth requirements. Each replica requires storage space to store a copy of the database, which can be a significant consideration for large databases with high data volumes. Additionally, efficient network connectivity is essential to ensure timely data synchronization between replicas, particularly in geographically dispersed environments.
6. **Single Point of Failure:** Database replication introduces the risk of a single point of failure. If the source database or a critical replica becomes unavailable, it can impact the availability and integrity of the entire replication system. Organizations need to implement redundancy and failover mechanisms to mitigate this risk and ensure high availability of replicated data.
7. **Replication Monitoring and Administration:** Replication systems require ongoing monitoring and administration to ensure their health and effectiveness. Monitoring replication status, detecting and resolving issues, managing replica synchronization, and performing regular maintenance tasks require dedicated resources and continuous attention.
8. **Data Security and Compliance:** Replication introduces additional considerations for data security and compliance. Organizations must ensure that replicated data remains protected and meets regulatory requirements. Implementing proper security measures, including access controls, encryption, and data privacy safeguards, becomes crucial to maintain data integrity and confidentiality.

9. Operational Complexity: Managing a replication environment with multiple replicas adds complexity to system operations and maintenance. Tasks such as adding new replicas, handling replica failures, or upgrading the replication infrastructure require careful planning and coordination.

Despite these limitations, database replication remains a valuable technique for achieving data availability, scalability, and fault tolerance in decentralized database management systems. By carefully considering these limitations and addressing them through appropriate design, configuration, and ongoing management, organizations can leverage the benefits of replication while mitigating potential challenges.

3.9 Strategies to implement

Direct mail, anti-entropy, and rumor mongering [20] are concepts and techniques related to data dissemination and synchronization in decentralized systems. These approaches play a significant role in maintaining data consistency, managing data updates, and ensuring information dissemination across multiple nodes or replicas. These techniques provide mechanisms for efficient data dissemination, synchronization, and maintaining data consistency in decentralized systems. Each technique offers unique advantages and considerations depending on the specific requirements and characteristics of the system. By leveraging these approaches appropriately, decentralized systems can ensure timely and consistent dissemination of updates, handle network disruptions or delays, and maintain data integrity across multiple nodes or replicas.

3.9.1 Direct Mail

Direct mail is a data dissemination technique commonly used in decentralized systems. In this approach, data updates or messages are directly sent from the source node to the target nodes that need to receive the information. The source node identifies the intended recipients and delivers the updates directly to them, bypassing intermediate nodes. This technique is efficient for targeted data dissemination,

as it minimizes unnecessary communication and reduces the dissemination delay. Direct mail is often used in scenarios where selective dissemination is required, such as sending updates to specific nodes or replicas.

3.9.2 Limitations

1. Scalability: Direct Mail can face challenges in highly scalable systems with a large number of nodes or replicas. As the number of recipients increases, the overhead of identifying and delivering updates to each recipient grows, potentially impacting performance and scalability.
2. Routing Efficiency: Efficient routing becomes crucial in Direct Mail to ensure updates reach their intended recipients. The design and implementation of efficient routing algorithms can be complex, especially when dealing with dynamic and changing network topologies.
3. Failure Handling: Direct Mail can be susceptible to node failures or network partitions. If a source node fails before delivering updates, or if a recipient node fails to receive updates, ensuring reliable delivery becomes challenging.

3.9.3 Complexities

1. Message Ordering: Maintaining the order of messages in Direct Mail can be challenging, especially when updates from different sources or on different paths arrive at recipients simultaneously. Implementing mechanisms to preserve the order of updates can introduce additional complexity.
2. Message Duplication and Overlapping: Direct Mail may result in message duplication or overlapping when multiple sources send updates to the same recipient or when multiple paths deliver updates concurrently. Handling and resolving such situations require careful consideration to avoid inconsistencies or unnecessary processing overhead.

3.9.4 Anti-Entropy

Anti-entropy is a technique used in distributed systems to ensure data consistency and synchronize updates across replicas. It addresses the problem of maintaining consistent replicas in the face of network partitions, failures, or delays. Anti-entropy involves periodically comparing and exchanging data between replicas to identify and reconcile any differences or inconsistencies. The replicas exchange information about the data they store, and if any disparities are detected, the missing or outdated data is transmitted to bring the replicas back into sync. This technique helps mitigate the effects of network delays or failures and ensures that replicas remain consistent over time.

3.9.5 Limitations

1. **Communication Overhead:** Anti-Entropy involves exchanging data between replicas to identify and reconcile differences. This process incurs communication overhead, particularly when dealing with large datasets or frequent updates. The amount of data transferred during synchronization can impact network bandwidth and performance.
2. **Convergence Time:** The time required for replicas to converge to a consistent state can vary depending on the frequency of updates, network conditions, and the number of replicas. Achieving global consistency may take time, and during this convergence period, replicas might temporarily have inconsistent views of the data.
3. **Resource Consumption:** The process of comparing and exchanging data in Anti-Entropy requires computational resources and storage space. For systems with limited resources, managing the overhead of Anti-Entropy can be challenging.

3.9.6 Complexities

1. Conflict Resolution: Resolving conflicts that arise when replicas have divergent updates can be complex. Determining the correct resolution strategy and ensuring consistency across replicas may require careful consideration of application-specific requirements and conflict resolution mechanisms.
2. Synchronization Frequency: Deciding on the frequency of Anti-Entropy synchronization is non-trivial. Synchronizing too frequently can result in increased communication overhead, while infrequent synchronization may lead to larger data disparities and longer convergence times.

3.9.7 Rumor Mongering

Rumor-mongering, also known as gossip-based communication, is a decentralized communication technique inspired by the spread of rumours in social networks. In this approach, nodes or replicas communicate with each other by randomly selecting and exchanging information. When a node receives new data or an update, it disseminates that information to a subset of its neighbouring nodes. The receiving nodes, in turn, continue the process by spreading the information to their neighbours.[21] This iterative process continues until the information reaches all or a majority of the nodes in the system. Rumor-mongering is resilient to network partitions, as it does not rely on strict paths of communication. It allows information to propagate in a decentralized and self-organizing manner, ensuring that updates reach all nodes eventually.

3.9.8 Limitations

Reliability and Completeness: Rumor mongering relies on the random selection of neighbours for information dissemination. While it provides probabilistic guarantees of message propagation, there is a small chance that some nodes may not receive certain updates. Achieving complete and reliable dissemination across all nodes is not guaranteed.

Convergence Time: The time taken for information to propagate to all nodes in a decentralized system using rumor-mongering can vary significantly. Factors such as network topology, node connectivity, and the rate of gossip exchanges influence the convergence time. It can take longer for updates to reach all nodes compared to other techniques.

Overhead and Redundancy: Rumor mongering involves frequent message exchanges between nodes, which can generate redundant network traffic.

3.9.9 Complexities

Network Partition Handling: When network partitions occur, rumor-mongering can lead to the dissemination of divergent updates across different partitions. Handling network partitions and resolving inconsistencies require additional mechanisms and protocols to ensure data consistency.

Gossip Protocol Design: Designing efficient and robust gossip protocols can be complex. Determining the parameters, such as gossiping rate, message forwarding strategy, and neighbour selection policies, requires careful consideration of the system's characteristics and goals.

4 Proposed Structure

Our proposed solution aims to address the challenges of data loss prevention and ensure data consistency in a decentralized database management system. We present a peer-to-peer gossip-based solution where each node in the database cluster acts as a witness for transactions and collaborates to reach a consensus on the current state of the database. To achieve this, we leverage the Raft consensus algorithm and the replicated log method. There are previous references of using consensus algorithm to prevent data loss which doesn't have the notion of database management [7] [22]

4.0.1 Replicated LogChain Method

In our solution, we proposed to the log to LogChain and replicating it maintains the integrity of the data which was done in the following way. A log has reference to the hash of the previous log which in turn has the hash of its previous log creating a LogChain and replicating this LogChain across the network for integrity [23].

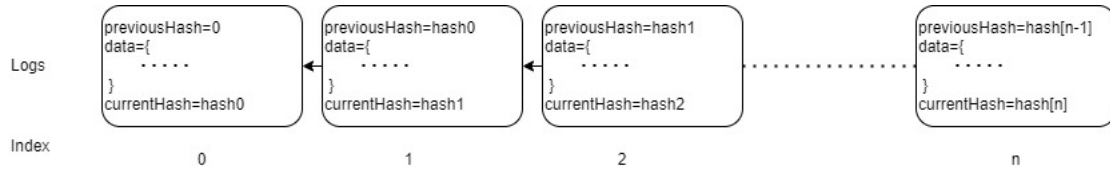


Figure 3: Editing the log to LogChain and replicating it maintains the integrity of the data which was done in the following way

4.0.2 Client Authenticity

Each client needs to have a registered public-private key pair with the cluster to interact. Every request sent from the client needs to be signed by the client which will be verified by the cluster before committing any transaction on its LogChain.

4.0.3 Reaching Consensus with Raft Algorithm Instead of Acknowledgement of Database Commit

To ensure consistency and agreement on the state of the database, we employ the Raft consensus algorithm. Raft provides a fault-tolerant and leader-based approach to consensus. Nodes participate in leader elections, log replication, and commit decisions to maintain a consistent and up-to-date state across the database cluster. The leader node coordinates the replication of the transaction logs and ensures that all nodes eventually reach a consensus on the committed transactions [24].

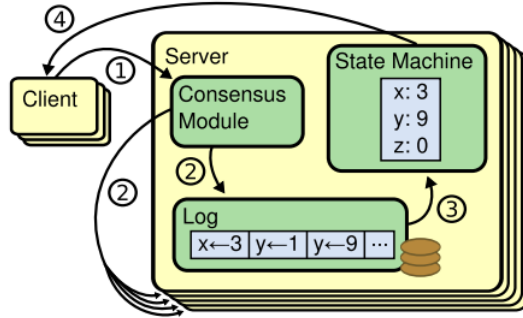


Figure 4: Replicated state machine architecture

4.1 Benefits of our solution

By combining the peer-to-peer gossip-based approach, Raft consensus algorithm, and replicated log method, our solution provides the following benefits:

- **Data Consistency:** The witness nodes and the consensus mechanism ensure that all nodes eventually agree on the state of the database, maintaining data consistency across the decentralized system.
- **Fault Tolerance:** The use of the Raft algorithm and replication of transaction logs provide fault tolerance capabilities. In the event of node failures or network disruptions, the system can continue to operate and reach a consensus by electing a new leader and replicating logs to the surviving nodes.

- Scalability: The gossip-based communication model and decentralized architecture allow for scalable growth of the database cluster. As the number of nodes increases, the system can handle a higher transaction load and view requests.
- Data Loss Prevention: By leveraging replication and consensus mechanisms, our solution reduces the risk of data loss. Each transaction is captured in the replicated logs, ensuring that even in the event of node failures or network issues, the data can be recovered and the system can maintain data integrity.

4.2 Proposed Architecture

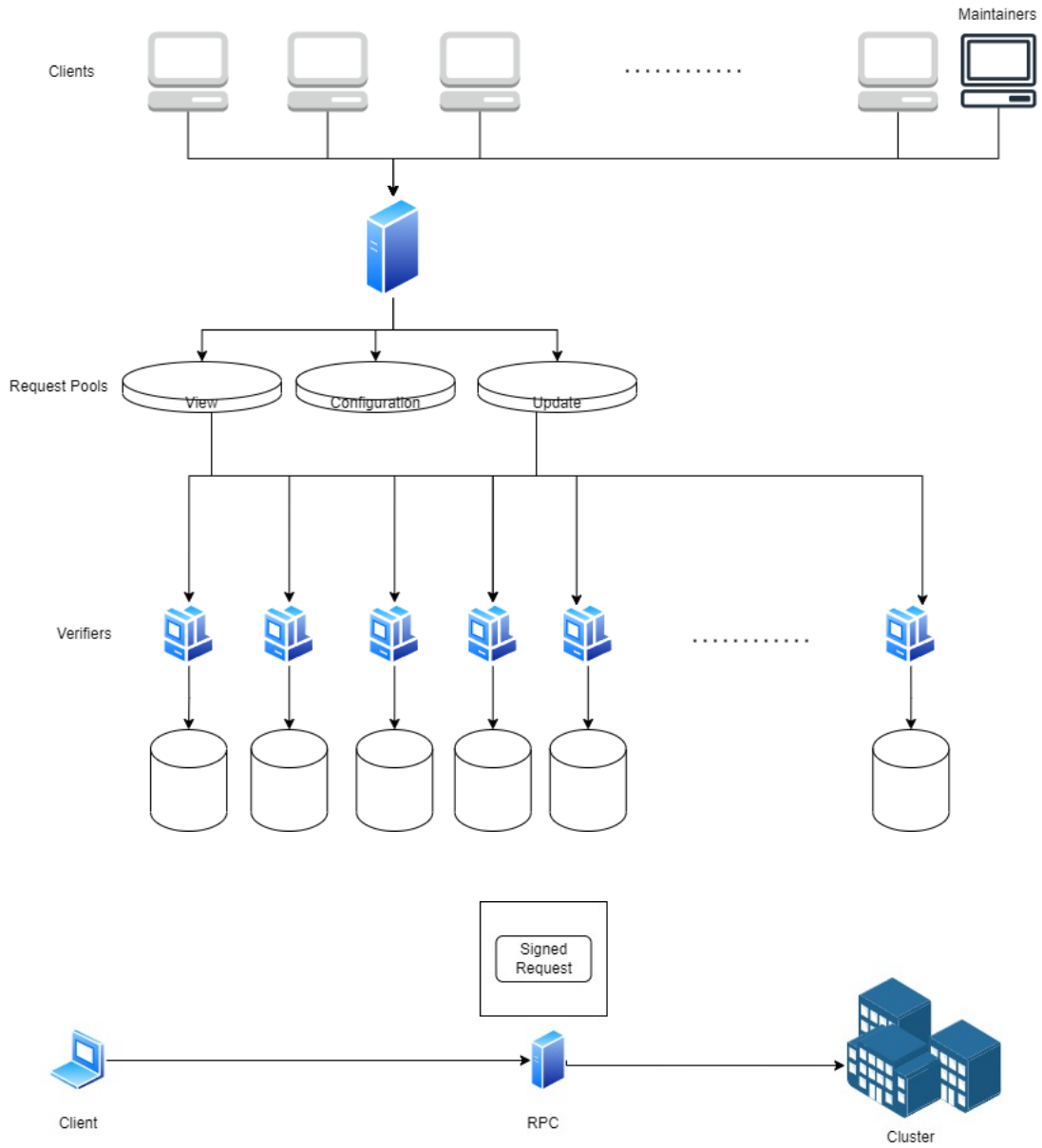


Figure 5: Higher Level Architecture of the Ecosystem

There are 4 types of machines included in this whole ecosystem

1. Client: Sends Transaction request or view request to the cluster
2. Verifier/Node: Constructs the cluster
3. Maintainers: Changes any configuration to the cluster e.g. adding a node or removing any node.
4. Entrypoint RPC: This is the entry point for any requests sent to the server

In addition, there are 3 types of request pools

1. Update: All the transactional queries are stored here
2. View: All the view queries are stored here
3. Configuration: All the maintainer's requests are stored here

Now we will discuss the execution flow of all the machines

4.3 Client/Maintainer

1. Client signs the request and sends it to the entry point RPC
2. RPC forwards the request to the cluster and the cluster verifies the signature
3. Client Receives a Response

4.4 Entrypoint RPC

1. Forwards the request to the cluster
2. Return the response to the client

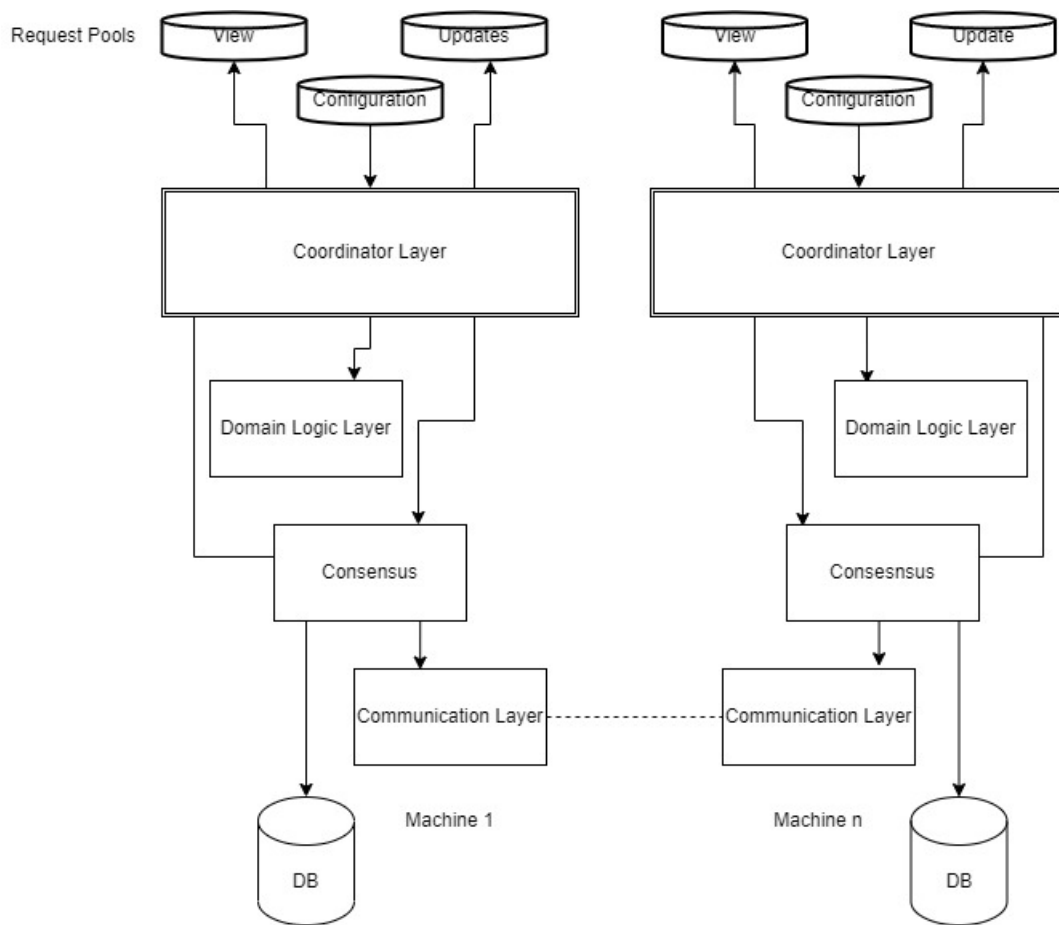


Figure 6: Node Architecture

4.5 Verifier/Node(Cluster)

1. Receives Update or View Request on the Request Pools
2. Coordination Layer uses the Domain Logic Layer to verify the signature
3. Gets the Last Log Hash
4. Appends the log hash with the current Log and calculates the new Log Hash
5. Reaches Consensus and Entries the updated Log into the LogChain

4.6 Execution Scenarios

1. Node Failure: In the case of one node failure the voting is initiated and a new leader is chosen

2. Node Append: Maintainer appends the node and the log is compressed and replicated so as the state machine
3. Malicious Activity: Client signatures are impossible to forge and changing anything in the hash chain requires extensive computation which is not feasible [23].

5 Simulation

5.1 Implementation Architecture

Communication Protocol: OQM Message Queue [25]

Consensus Algorithm: Raft Consensus Algorithm [26]

Database Construction: Replicated LogChain Method

Authentication: Private-Public Key Cryptography

Development Language: Node.Js

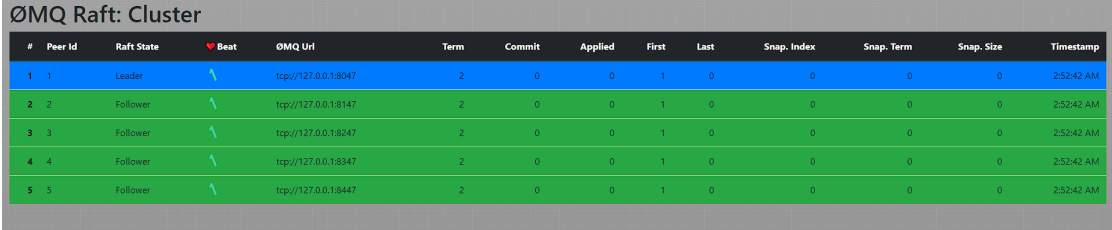
5.2 simulation results

We ran the simulation using node-zmq-raft instantiating 5 virtual state machines in the same computer with the configuration:

Processor: Ryzen 7 5700g

RAM: 16GB

We ran the simulation for 1000 transactions and the mean-time is given below:



#	Peer Id	Raft State	Beat	ØMQ Url	Term	Commit	Applied	First	Last	Snap. Index	Snap. Term	Snap. Size	Timestamp
1	1	Leader	✓	tcp://127.0.0.1:8047	2	0	0	1	0	0	0	0	2:52:42 AM
2	2	Follower	✓	tcp://127.0.0.1:8147	2	0	0	1	0	0	0	0	2:52:42 AM
3	3	Follower	✓	tcp://127.0.0.1:8247	2	0	0	1	0	0	0	0	2:52:42 AM
4	4	Follower	✓	tcp://127.0.0.1:8347	2	0	0	1	0	0	0	0	2:52:42 AM
5	5	Follower	✓	tcp://127.0.0.1:8447	2	0	0	1	0	0	0	0	2:52:42 AM

Figure 7: Heartbeat of five virtual state machines

$\mu_1 = \text{Execution Time of 1000 transactions (without database operation)} = 398.705$
ms

$\mu_2 = \text{Execution Time of 1000 transactions(with database operation)} = 6.585$ s

6 Result Analysis

The formula for calculating the execution time is:

$$T = T_c \times (\lambda \times n \times (n_t) + (\lambda_d) \times (n_t)) \text{ where,}$$

T = Total Time

T_c = Optimistic Consensus Time

λ = Network Propagation Delay

λ_d = Database Operational Delay

n_t = Number of Transactions

n = Number of nodes

Now, for the first case $\lambda_d = 0$, $\lambda = 1$ and $n_t = 1000$, we can get the T_c value and divide it by 5 (as all virtual machines are run in the same machine). We get the $T_c = 0.02$ ms.

Using this, we get the $\lambda_d = 60$ ms (in practice which is very low. Due to our implementation limitations this number is a lot high for one transaction committed onto the database).

Let us consider $T = 1000$ ms (1 second). Using the above values we can get the number of transactions in 1 second. We calculated that

if $\lambda = 1$ ms (Same Physical Network) - Transaction Speed = 770 tps

if $\lambda = 20$ ms - Transaction Speed = 312 tps

As the number of machines involved in executing a query increases, the time required for query execution also tends to increase. This is primarily due to two factors: propagation delay and consensus time associated with each machine. Propagation delay refers to the time it takes for information to travel between different machines in the system, influenced by factors such as network latency and the physical distance between machines. As the number of machines increases,

the overall communication overhead rises, resulting in longer propagation delays and subsequently prolonging the query execution time.

Another significant factor impacting execution time is the consensus time required to reach an agreement among the replicated databases. Consensus protocols, such as the popular Raft or Paxos algorithms, are often employed to ensure consistency across replicas. However, as the number of participating machines grows, the process of reaching consensus becomes more time-consuming. Each machine needs to communicate, exchange information, and arrive at a consensus, which can introduce additional delays in the query processing phase.

Despite the potential benefits of having a larger number of replicated databases, practical considerations, especially in terms of cost, make it inefficient to have more than a certain threshold, typically around 10, in most scenarios. The expenses associated with maintaining and managing a large number of replicas, including hardware costs, maintenance efforts, and operational overhead, can become prohibitive. Therefore, in real-world scenarios, it is often impractical to go beyond a certain number of replicated databases due to cost constraints.

It is important to note that the simulation mentioned in the context was conducted in a sandbox environment, which provides a controlled and isolated setting for testing purposes. While the results obtained from the simulation can offer valuable insights, it is crucial to acknowledge that real-world production environments introduce additional complexities and variables that can affect the outcome. Factors such as varying network conditions, hardware limitations, and system load can significantly impact system performance and consequently influence the results in a production setting. Thus, conducting further experiments and evaluations in an actual production environment would be necessary to obtain a more accurate understanding of the system's behavior and performance in practical scenarios.

7 Future Works

The process of updating the state machine in each machine is notably inefficient, primarily because the state machines employed are in the form of databases and involve significant transaction time delays. To address this inefficiency, we can draw inspiration from the concept of Chain replication, as described in the work by Fritchie et al. [27]. By leveraging the insights and techniques presented in that study, we can explore alternative approaches to updating the state of the database that are more efficient and cost-effective compared to traditional transactions.

It is worth noting that the transactional nature of updating the state machines introduces a substantial overhead, both in terms of time and resources. Transactions often involve multiple steps, such as acquiring locks, validating data integrity, and committing changes, which can be computationally expensive. By rethinking the update process and finding ways to minimize or eliminate the need for expensive transactions, we can significantly improve the efficiency of state updates in each machine.

Furthermore, it is crucial to address issues related to inefficient data caching in the databases, as highlighted in the study by Papaioannou et al. [28]. Inefficient data caching can negatively impact the read speed of the databases, leading to delays in accessing and retrieving data. By focusing on optimizing data caching strategies and addressing the identified inefficiencies, we can enhance the overall read performance of the databases and mitigate the issues associated with data access.

To summarize, to improve the efficiency of state-machine updates in each machine, we can explore alternative methods inspired by Chain replication. Additionally, addressing inefficiencies in data caching can help improve the read speed of databases. By leveraging these approaches, we can work towards a more efficient and optimized system for updating and accessing the state of the database.

8 Conclusion

The utilization of the Raft Consensus Algorithm introduces a highly innovative and pioneering approach in the development of decentralized systems. This algorithm serves as a fundamental methodology for constructing a network of nodes that operate in a distributed and coordinated manner. The result is a system that not only showcases impressive qualities such as resilience, scalability, and consistency, but also ensures these attributes extend throughout the entire cluster network.

One of the primary challenges faced in designing decentralized systems is the achievement of consistent performance. Previous methods, such as the primary-backup techniques or chain replication approach, have showcased limitations in this regard. Specifically, these conventional methods often sacrifice the inherent transaction speed of the native database, thereby impeding the system's overall performance.

In contrast, the Raft Consensus Algorithm provides a solution to this problem. By leveraging a leader-based approach and employing a replicated log as the primary data structure, the algorithm ensures that the system maintains a high level of consistency while preserving the transaction speed of the underlying database. This unique characteristic sets it apart from traditional methods and establishes it as a superior choice for constructing decentralized database systems.

In summary, the Raft Consensus Algorithm revolutionizes the construction of decentralized database systems by offering a robust and efficient solution. Its ability to maintain resilience, scalability, and consistency throughout the cluster network, coupled with its capacity to preserve transaction speed, makes it a truly exceptional approach for building decentralized systems in a wide range of applications.

References

- [1] S. Liu and R. Kuhn, “Data loss prevention,” *IT professional*, vol. 12, no. 2, pp. 10–13, 2010.
- [2] L. Cheng, F. Liu, and D. Yao, “Enterprise data breach: Causes, challenges, prevention, and future directions,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 5, e1211, 2017.
- [3] R. Van Renesse and F. B. Schneider, “Chain replication for supporting high throughput and availability.,” in *OSDI*, vol. 4, 2004.
- [4] B. Calder, J. Wang, A. Ogus, *et al.*, “Windows azure storage: A highly available cloud storage service with strong consistency,” in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 143–157.
- [5] A. Adya, W. J. Bolosky, M. Castro, *et al.*, “Farsite: Federated, available, and reliable storage for an incompletely trusted environment,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 1–14, 2002.
- [6] F. B. Schneider, “Implementing fault-tolerant services using the state machine approach: A tutorial,” *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.
- [7] V. Iancu and I. Ignat, “A peer-to-peer consensus algorithm to enable storage reliability for a decentralized distributed database,” in *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, vol. 2, 2010, pp. 1–6. DOI: 10.1109/AQTR.2010.5520830.
- [8] M. K. Lai and K. Schildkamp, “Data-based decision making: An overview,” *Data-based decision making in education: Challenges and opportunities*, pp. 9–21, 2013.
- [9] R. S. Sandhu, “On five definitions of data integrity.,” in *DBSec*, Citeseer, 1993, pp. 257–267.

- [10] G. Sivathanu, C. P. Wright, and E. Zadok, “Ensuring data integrity in storage: Techniques and applications,” in *Proceedings of the 2005 ACM workshop on Storage security and survivability*, 2005, pp. 26–36.
- [11] G. J. Matthews and O. Harel, “Data confidentiality: A review of methods for statistical disclosure limitation and methods for assessing privacy,” 2011.
- [12] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, “Blockchain based data integrity service framework for iot data,” in *2017 IEEE International Conference on Web Services (ICWS)*, IEEE, 2017, pp. 468–475.
- [13] S. M. Diesburg and A.-I. A. Wang, “A survey of confidential data storage and deletion methods,” *ACM Computing Surveys (CSUR)*, vol. 43, no. 1, pp. 1–37, 2010.
- [14] Z. Zhao, H. Zhao, Q. Zhuang, *et al.*, “Efficiently supporting multi-level serializability in decentralized database systems,” *IEEE Transactions on Knowledge and Data Engineering*, 2023.
- [15] D. Serrano, M. Patiño-Martinez, R. Jiménez-Peris, and B. Kemme, “Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation,” in *13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)*, IEEE, 2007, pp. 290–297.
- [16] F. Cristian, “Understanding fault-tolerant distributed systems,” *Communications of the ACM*, vol. 34, no. 2, pp. 56–78, 1991.
- [17] H. Lamahemedi, B. Szymanski, Z. Shentu, and E. Deelman, “Data replication strategies in grid environments,” in *Fifth International Conference on Algorithms and Architectures for Parallel Processing, 2002. Proceedings.*, IEEE, 2002, pp. 378–383.
- [18] S. Suguna and A. Suhasini, “Overview of data backup and disaster recovery in cloud,” in *International Conference on Information Communication and Embedded Systems (ICICES2014)*, IEEE, 2014, pp. 1–7.

- [19] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, “Database replication techniques: A three parameter classification,” in *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000*, IEEE, 2000, pp. 206–215.
- [20] J. Holliday, R. Steinke, D. Agrawal, and A. El Abbadi, “Epidemic algorithms for replicated databases,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1218–1238, 2003.
- [21] S. Deb, M. Médard, and C. Choute, “Algebraic gossip: A network coding approach to optimal multiple rumor mongering,” *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2486–2507, 2006.
- [22] R. Jiménez-Peris, M. Patiño-Martinez, G. Alonso, and B. Kemme, “Are quorums an alternative for data replication?” *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 3, pp. 257–294, 2003.
- [23] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Decentralized business review*, p. 21 260, 2008.
- [24] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 305–319.
- [25] M. Sústrik *et al.*, “Zeromq,” *Introduction Amy Brown and Greg Wilson*, 2015.
- [26] L. H. royaltm dependabot[bot] dependabot[bot], *An opinionated raft implementation powered by ømq*, <https://github.com/royaltm/node-zmq-raft>, 2017.
- [27] S. L. Fritchie, “Chain replication in theory and in practice,” in *Proceedings of the 9th ACM SIGPLAN workshop on Erlang*, 2010, pp. 33–44.
- [28] A. Papaioannou and K. Magoutis, “Addressing the read-performance impact of reconfigurations in replicated key-value stores,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 9, pp. 2106–2119, 2021.