Distributed Systems and Databases: A Survey on Key Concepts and Technologies

Abstract

This survey paper provides a comprehensive examination of key concepts and technologies underpinning distributed systems and databases. It delves into the CAP theorem, ACID and BASE models, consensus algorithms, and consistency models, each crucial for ensuring reliability and scalability in distributed environments. The CAP theorem highlights the trade-offs between consistency, availability, and partition tolerance, guiding architectural decisions. ACID and BASE models offer contrasting paradigms for transaction management, influencing the balance between consistency and availability. Consensus algorithms like Paxos and Raft are pivotal for achieving agreement across distributed nodes, despite challenges in performance and fault tolerance. Innovations such as Egalitarian Paxos (EPaxos) enhance scalability but may introduce safety violations under certain conditions. The transition from NoSQL to NewSQL databases, exemplified by Google Spanner, addresses modern data-intensive application needs by combining scalability with strong consistency. Sharding and advanced spanner constructions are critical for managing complexity and enhancing system performance and fault tolerance. Consistency models, including linearizability and eventual consistency, offer varying levels of data integrity and availability, tailored to specific application requirements. The survey underscores the importance of ongoing research and innovation in addressing emerging challenges and optimizing distributed systems' performance. Future research directions include developing programming languages for monotonic distributed systems and advancing polyglot data management frameworks. Understanding these foundational concepts is essential for developing robust, scalable distributed systems capable of adapting to evolving computing demands.

1 Introduction

1.1 Importance of Distributed Systems and Databases

Distributed systems and databases form the backbone of modern computing, enabling efficient management and processing of large-scale data. Their significance is evident in their ability to maintain reliability and consistency across diverse applications, particularly in cloud environments where integrity verification is crucial due to the involvement of multiple clients and potential threats from malicious cloud providers [1, 2]. The evolution of distributed databases is largely influenced by the challenges of adaptivity and finality in blockchain protocols, underscoring their importance in decentralized settings [3]. Moreover, these systems play a critical role in defining concurrent behaviors, especially in large-scale data processing scenarios [4]. The persistence of data in concurrent structures poses significant challenges, often relying on blocking mechanisms that hinder progress, thus necessitating the development of nonblocking approaches to advance the persistence frontier [5].

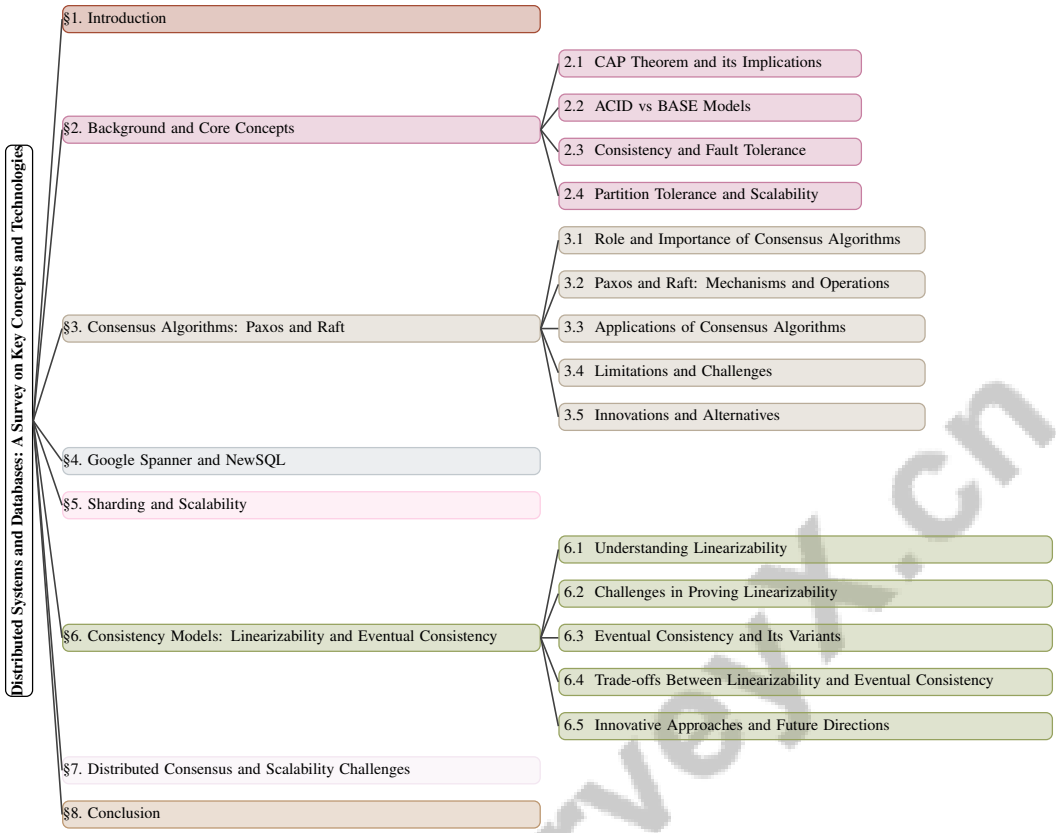


Figure 1: chapter structure

1.2 Key Concepts and Technologies Overview

Key concepts and technologies underpinning distributed systems and databases are essential for addressing the challenges of contemporary computing environments. The CAP theorem provides a foundational framework for understanding the trade-offs among consistency, availability, and partition tolerance critical for distributed database design [6]. Complementarily, the ACID and BASE models illustrate a spectrum of transaction management approaches that ensure reliability within database systems [7].

The advent of evolutionary data systems (EDS) represents a notable advancement, allowing systems to adapt their architecture to evolving workloads, thereby overcoming the limitations of static systems [8]. Polyglot data management leverages various data stores to address diverse use cases effectively [9], while polystore databases integrate SQL, NoSQL, and NewSQL to optimize data exchange and query execution across heterogeneous environments [10].

Consensus algorithms such as Paxos and Raft are pivotal for ensuring agreement on a single data value across distributed nodes, thus maintaining consistency. These mechanisms extend to blockchain technologies, which offer decentralized architectures and consensus models that help mitigate operational costs in applications like mobile peer-to-peer games [11]. Additionally, strong linearizability is explored for its implications in concurrent programs, particularly in preserving hyper-properties through consensus primitives [12].

The complexities arising from network partitions and failures necessitate new abstractions and models, including fork-linearizability and the Conflict-free Operation verification Protocol (COP) for consistency verification in remote, untrusted environments [13, 14]. The Matrix Event Graph (MEG) is highlighted as a replicated data type that enhances resilience and scalability in decentralized applications [15]. These foundational concepts and technologies will be examined in detail in subsequent sections, elucidating their roles and implications in distributed systems and databases.

AI-generated, for reference only.

1.3 Structure of the Survey

This survey is systematically organized to provide an in-depth exploration of distributed systems and databases, emphasizing key concepts and technologies that define this field. The paper begins with a comprehensive overview of the significance of distributed systems and databases in modern computing, highlighting their essential roles in ensuring data consistency, availability, and fault tolerance. This foundation supports a thorough examination of concepts such as traditional relational databases, NoSQL, and emerging NewSQL systems, alongside the implications of the CAP theorem and its alternatives for system design and performance in large-scale applications [16, 7, 17, 18, 19].

The survey then delves into background and core concepts, providing detailed discussions on the CAP theorem, ACID and BASE models, and their implications for database transactions, as well as the significance of consistency, availability, and partition tolerance. Subsequent sections focus on consensus algorithms, particularly Paxos and Raft, exploring their mechanisms, applications, and limitations.

The discussion extends to Google Spanner and NewSQL databases, detailing their characteristics and the transition from NoSQL to NewSQL. Sharding and scalability are also addressed, illustrating how these concepts enhance the performance and reliability of distributed systems. The survey compares consistency models such as linearizability and eventual consistency, analyzing their trade-offs and applications. Finally, it examines distributed consensus and scalability challenges, offering insights into innovative algorithms and strategies to overcome these issues. Each section builds upon the previous one, providing a comprehensive understanding of the intricacies and advancements in distributed systems and databases, as outlined in the framework proposed by [4].The following sections are organized as shown in Figure 1.

2 Background and Core Concepts

2.1 CAP Theorem and its Implications

The CAP theorem, introduced by Eric Brewer, is pivotal in understanding distributed databases, emphasizing the inherent trade-offs among Consistency, Availability, and Partition Tolerance [18]. It posits that a distributed system cannot simultaneously achieve all three properties, necessitating a prioritization based on application requirements [3]. Consistency ensures uniform data state across nodes, crucial for data integrity amid concurrent modifications [20]. However, maintaining strict consistency can undermine availability and partition tolerance [18].

Availability ensures a system's responsiveness, even during network partitions when nodes may be isolated [9]. Partition tolerance is essential in environments susceptible to network failures, allowing continued operation despite disruptions [21]. Consequently, systems often adopt models like BASE (Basically Available, Soft state, Eventually consistent) when eventual consistency is adequate [2].

The CAP theorem also influences consensus algorithms and replication protocols, particularly in maintaining linearizability amid Byzantine failures [20]. It guides the design of replication protocols balancing throughput and latency, especially in inter-data center communications [21]. Recent critiques and expansions of the CAP theorem, including the CALM theorem, suggest that monotonic programs can simultaneously achieve consistency, availability, and partition tolerance, offering new perspectives on trade-offs in distributed systems [1]. These concepts continue to shape the theoretical and practical landscape of distributed databases [22].

2.2 ACID vs BASE Models

The ACID (Atomicity, Consistency, Isolation, Durability) and BASE (Basically Available, Soft state, Eventually consistent) models represent divergent approaches to database transaction management, each influencing distributed systems' reliability and performance. ACID transactions provide robust guarantees, ensuring all operations within a transaction succeed or fail collectively, critical for applications requiring strict consistency [23]. Conversely, BASE prioritizes availability and partition tolerance, accepting temporary inconsistencies, making it suitable for systems where high availability is paramount [24].

The ACID-BASE dichotomy reflects broader trade-offs between consistency and availability. ACID's stringent guarantees can lead to performance overheads in distributed environments [18], while BASE's focus on availability supports scalable, resilient systems at the potential cost of temporary inconsistencies [9]. Recent efforts aim to blend these models' strengths, exploring checkpointing and rollback algorithms to balance correctness and performance [25]. The CALM theorem further suggests that monotonic programs can achieve consistency, availability, and partition tolerance, enriching the understanding of distributed systems' limitations and possibilities [1].

Choosing between ACID and BASE models depends on application-specific needs. Systems requiring immediate consistency may favor ACID, while those prioritizing scalability and availability might opt for BASE. Understanding these models' principles is crucial for designing distributed databases that balance competing demands [18].

2.3 Consistency and Fault Tolerance

Consistency and fault tolerance are vital in distributed systems, ensuring reliable operation despite failures and network disruptions. Consistency ensures uniform data state across nodes, crucial in concurrent environments [26]. Advanced models like distributional and fragmented linearizability enhance concurrent data structures' performance under high concurrency [20].

Fault tolerance enables systems to function correctly despite failures, such as node crashes or Byzantine faults [2]. Byzantine faults necessitate robust consistency models to maintain reliability. Traditional models often falter under crash-failure conditions, highlighting the need for advanced mechanisms [27].

The interplay between consistency and fault tolerance is complex, as high consistency requires synchronization, affecting performance in high-latency environments [16]. Challenges include stable participant sets in unstructured networks and guaranteeing consistency with low latency in data centers [28, 25]. Innovative approaches like the SWARM model minimize roundtrips, reducing latency while maintaining consistency [5]. Data replication ensures reliability, availability, and fault tolerance, supporting consistent update propagation across nodes [1].

Consistency and fault tolerance are foundational for distributed systems, balancing synchronization, concurrency, and adversarial conditions to maintain reliable operations. Approaches like Just-Right Consistency and the CALM theorem provide insights into achieving consistency without excessive synchronization, enabling robust performance even amid network partitioning [29, 30, 25, 1, 31].

2.4 Partition Tolerance and Scalability

Partition tolerance is crucial for distributed systems, allowing correct operation despite network partitions or failures, essential for large-scale systems where disruptions are inevitable [18]. This capability supports scalability, enabling systems to manage increased loads and expand across diverse networks without performance degradation.

Scalability is achieved through data structures and algorithms optimizing communication and computation amid partitions. Systems like nbMontage convert nonblocking concurrent data structures into persistent ones, enhancing scalability [5]. The congested clique model efficiently constructs spanners, optimizing communication [18]. These models reduce communication rounds, improving scalability and managing increased loads effectively.

Scalability involves reconciling mathematical foundations and operational semantics across databases, which can cause bottlenecks [32]. In distributed file systems, centralized metadata management can hinder performance as data volume grows, necessitating scalable solutions [33]. Categorizing methods based on delay sensitivity provides insights into optimizing scalability under varying conditions [18].

Partition tolerance is integral to scalability, with innovative algorithms and data structures enhancing fault tolerance and adapting to network conditions. This approach balances consistency and availability, as highlighted by delay-sensitivity models and the CAL theorem, addressing challenges posed by network disruptions [16, 18].

Category	Feature	Method
Role and Importance of Consensus Algorithms	Data Integrity and Verification Concurrency and Performance Fault Tolerance and Reliability	VICOS[2], ABV[28] nbM[5] CLC[3]
Paxos and Raft: Mechanisms and Operations	Consensus Strategies Data Replication Techniques	N/A[34] HMS[35]
Applications of Consensus Algorithms	Scheduled Validation	PoT[11]
Limitations and Challenges	Communication Complexity Efficiency in Distributed Systems Consistency and Atomicity	LIC[36], CCSC[37] N/A[38], Crisis[39] NBTC[40]
Innovations and Alternatives	Adaptive Strategies	EFP[41]

Table 1: This table provides a comprehensive overview of various consensus algorithms, highlighting their roles, features, and methods. It categorizes the algorithms based on their importance, mechanisms, applications, limitations, and innovations, offering insights into their operational dynamics and challenges in distributed systems.

3 Consensus Algorithms: Paxos and Raft

Consensus algorithms are pivotal for ensuring reliability and consistency in distributed systems, addressing CAP theorem trade-offs between consistency, availability, and partition tolerance. While permissionless blockchains use resource-intensive proof-of-work, permissioned blockchains favor efficient algorithms like Proof-of-Authority and Practical Byzantine Fault Tolerance, each offering different security and performance levels. Emerging frameworks, such as DAG-based consensus protocols, illustrate diverse approaches in decentralized networks [42, 16, 39, 43, 18]. Paxos and Raft are prominent algorithms, each with unique mechanisms and characteristics crucial for system integrity and performance. This section explores the role of consensus algorithms and examines Paxos and Raft's mechanisms. Table **??** provides a comparative analysis of the Paxos and Raft consensus algorithms, detailing their distinct operational mechanisms and complexity levels. Additionally, Table 1 presents a detailed categorization of consensus algorithms, elucidating their significance, operational mechanisms, applications, and associated challenges in distributed systems.

3.1 Role and Importance of Consensus Algorithms

Method Name	**Data Consistency**	**Performance Optimization**	**Application Domains**
VICOS[2]	Fork-linearizability	Lower Communication Overhead	Cloud Object Storage
ABV[28]	Weak Consistency Guarantees	Balance Correctness Efficiency	Geo-replicated Systems
nbM[5]	Buffered Durable Linearizability	High Performance	Concurrent Data Structures
CLC[3]	Byzantine Fault Tolerant	Transaction Finality Time	Blockchain Protocols

Table 2: Comparison of consensus algorithms across various dimensions, highlighting their data consistency models, performance optimization strategies, and application domains. The table provides insights into the trade-offs and specific uses of each method in distributed systems, including cloud storage, geo-replicated systems, concurrent data structures, and blockchain protocols.

Consensus algorithms are essential for achieving agreement on a single data value across nodes, crucial for data consistency and system reliability in distributed systems. Table 2 presents a comprehensive comparison of different consensus algorithms, emphasizing their importance in achieving data consistency and performance optimization across various application domains in distributed systems. They are vital in environments with potential node failures or malicious behavior, ensuring a consistent state [16]. Balancing latency and consistency is critical for maintaining integrity and efficiency, especially under unpredictable network conditions. In cloud storage, verification protocols like VICOS allow clients to verify data integrity without full server trust [2].

Developing semantics and specifications for concurrent libraries in weakly-consistent settings underscores the necessity of consensus algorithms for correct operation [28]. Frameworks on polyglot data management highlight the diverse roles of consensus algorithms [9]. Innovations such as nbMontage demonstrate nonblocking data structures' advantages over traditional locking, enhancing performance and resilience [5]. In blockchain technologies, consensus algorithms are critical, as seen in the checkpointed longest chain protocol [3].

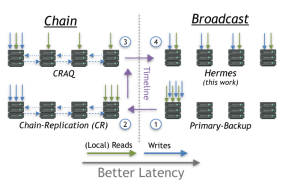
3.2 Paxos and Raft: Mechanisms and Operations

Paxos and Raft are renowned consensus algorithms ensuring nodes agree on a single data value, maintaining system consistency and reliability. They manage failures and network partitions using advanced techniques. The Crisis framework introduces probabilistically converging total order algorithms resilient in high-entropy networks, achieving near-optimal communication efficiency [39]. In permissioned blockchains, Proof-of-Authority and Practical Byzantine Fault Tolerance (PBFT) offer varying consistency and performance levels [42].

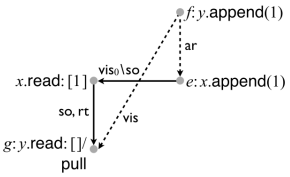
Paxos, introduced by Leslie Lamport, uses proposing, voting, and committing phases among nodes. A proposer suggests a value to a quorum of acceptors; if a majority agree, the value is chosen and communicated to learners, ensuring convergence. Paxos is fault-tolerant but criticized for complexity and performance overhead [44]. Egalitarian Paxos (EPaxos) reduces leader election needs, enhancing safety and consistency [44].

Raft, a more understandable alternative to Paxos, centers around a strong leader model. A leader manages log replication and ensures consistency across followers, simplifying coordination. The leader handles client requests, appends entries to its log, replicates them to followers, and commits entries once a majority acknowledge them. Raft emphasizes understandability and ease of implementation [45]. It integrates detectable operations into persistent lock-free data structures, enhancing recovery efficiency and consistency [46].

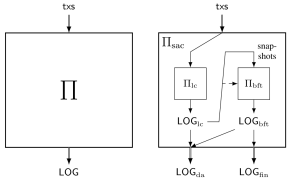
Both algorithms aim for consensus but differ in approaches. Paxos's decentralized model allows multiple proposers and acceptors, promoting resilience, while Raft's centralized approach with a leader simplifies coordination but may introduce vulnerabilities if the leader fails. This distinction highlights trade-offs between decentralization and efficiency, influencing performance and suitability [44, 42, 47, 38]. Evaluating these algorithms reveals their strengths and limitations. Hermes achieves high throughput and low latency while providing linearizable operations [35]. Adaptations of etcd implementing causal consistency with CRDTs enhance performance in edge environments [34].



(a) Chain-Based Data Replication with Hermes[35]



(b) A Graphical Representation of a Program Flow[48]



(c) The image shows a block diagram representing a system with two main components: a central square labeled "" and two smaller boxes labeled "sac" and "bft."[41]

Figure 2: Examples of Paxos and Raft: Mechanisms and Operations

As shown in Figure 2, consensus algorithms ensure nodes agree on a single data value, critical for consistency and reliability in distributed systems. Paxos and Raft exemplify this, each managing distributed environments' complexities. The first image illustrates "Chain-Based Data Replication with Hermes," showcasing a novel data replication approach enhancing fault tolerance and linearizability. The second image provides "A Graphical Representation of a Program Flow," emphasizing execution dynamics within a directed graph framework. Lastly, the third image presents a block diagram of a system architecture, depicting the interplay of components. Together, these visuals underscore the sophisticated mechanisms underpinning Paxos and Raft, highlighting their operational intricacies in achieving consensus.

3.3 Applications of Consensus Algorithms

Consensus algorithms are integral to distributed systems, ensuring agreement among nodes and data consistency across applications. They address CAP theorem trade-offs and implement notions like interval-linearizability [4, 16]. In blockchain technologies, they maintain decentralized, tamper-proof

ledgers. The Proof of Turn (PoT) mechanism exemplifies this by reducing server costs and ensuring fair gaming environments [11].

In distributed databases, consensus algorithms coordinate transactions and maintain data integrity across nodes. Achieving consensus ensures high availability and fault tolerance, maintaining consistent states despite network partitions or failures. In permissioned blockchains, the choice of consensus algorithm impacts security and performance; some, like Proof-of-Authority, may struggle with consistency under certain conditions, while others, such as Practical Byzantine Fault Tolerance, offer better resilience at a performance cost [42, 16]. Consensus algorithms enable reliable data replication and synchronization, allowing applications to scale while maintaining consistency.

They are also applied in verifying distributed transactions, facilitating the analysis of transaction histories to ensure atomicity and correctness. Techniques like Forward and Backward Zone (FZF) analysis enhance verification efficiency [49]. Additionally, consensus algorithms in cloud storage systems ensure data consistency and integrity across data centers. Mechanisms like CRDTs and Transactional Causal Consistency allow asynchronous updates while maintaining correctness. Protocols like VICOS enhance data integrity, enabling clients to detect discrepancies in data stored on potentially untrustworthy cloud services [2, 31, 14].

As shown in Figure 3, this figure illustrates the applications of consensus algorithms across three primary domains: blockchain technologies, distributed databases, and cloud storage systems. Each domain showcases unique implementations and benefits, such as decentralized ledgers, transaction coordination, and data consistency, emphasizing the critical role of consensus algorithms in maintaining system reliability and integrity. The practical applications of consensus algorithms like Paxos and Raft reveal their foundational concepts and operational dynamics within complex systems. The illustrations begin with a flowchart delineating the relationships between various theorems and concepts, reflecting the structured nature of consensus algorithms. The depiction of interconnected nodes emphasizes the layered architecture of systems relying on these algorithms, showcasing their ability to manage complex interactions. Lastly, the network of interconnected trees illustrates how consensus algorithms maintain order within distributed networks. Together, these visuals underscore the critical role of consensus algorithms in ensuring consistency and reliability across applications.

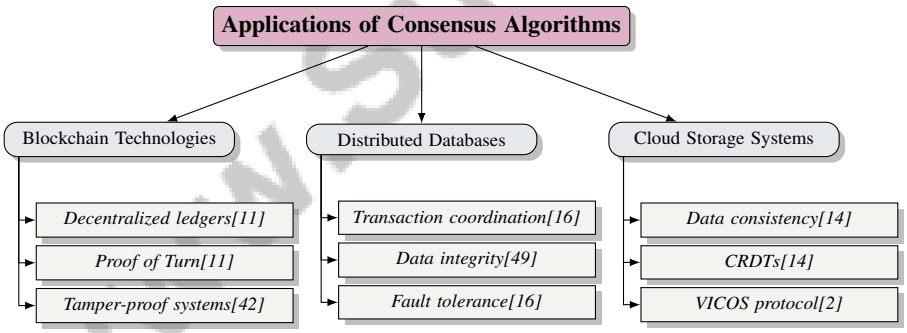


Figure 3: This figure illustrates the applications of consensus algorithms across three primary domains: blockchain technologies, distributed databases, and cloud storage systems. Each domain showcases unique implementations and benefits, such as decentralized ledgers, transaction coordination, and data consistency, emphasizing the critical role of consensus algorithms in maintaining system reliability and integrity.

3.4 Limitations and Challenges

Implementing consensus algorithms like Paxos and Raft in distributed systems presents challenges regarding performance, consistency, and fault tolerance. Achieving resilience against Byzantine failures is a significant challenge, requiring complex mechanisms to ensure consistency and security [50]. Though Paxos and Raft manage node failures and network partitions, they struggle under extreme conditions or high partitioning levels, where maintaining consistency and availability becomes difficult. Their performance hinges on leader-based architectures, which can become bottlenecks in high-load scenarios. Nezha addresses this by separating execution from commitment, improving performance [38].

Constructing spanner algorithms is challenging, as achieving time- and communication-optimality is difficult. This limitation affects consensus algorithms' efficiency, relying on effective communication and coordination [36]. Existing methods highlight the challenges in achieving efficient consensus and scalability, particularly in high communication overhead environments [37]. Proof-of-Authority algorithms underscore trade-offs between performance and consistency guarantees, performing well in low-latency environments but lacking robust consistency under adversarial conditions [42].

Achieving total order in dynamic environments requires balancing communication overhead and fault tolerance. Crisis offers a robust mechanism for achieving total order with lower communication overhead but faces challenges in maintaining fault tolerance [39]. Identifying linearization points in complex data structures and ensuring atomic execution complicates consensus algorithms' implementation, necessitating advanced techniques for managing consistency and performance [40]. The trade-off in permissionless blockchain protocols, where systems can be adaptive or have finality but not both, further complicates the distributed consensus landscape [47].

3.5 Innovations and Alternatives

Recent advancements in consensus algorithms introduce innovative approaches and alternatives to traditional models like Paxos and Raft, enhancing performance, resilience, and adaptability. Ebb-and-flow protocols dynamically adjust operational parameters based on network conditions and adversarial strategies, providing availability and finality [41]. Future research could focus on enhancing these protocols' resilience against sophisticated adversarial strategies, integrating them into existing blockchain frameworks for more robust systems.

Alternative approaches explore hybrid models combining elements of Paxos and Raft, optimizing performance and reliability. These models balance Paxos's decentralized nature with Raft's leader-based efficiency, providing flexible and scalable solutions. Integrating features like dynamic leader election and adaptive quorum sizes can improve fault tolerance and minimize latency, enhancing permissioned blockchains' reliability in real-world applications [42, 45].

In exploring the advancements in database technologies, particularly the evolution from NoSQL to NewSQL, it is essential to understand the underlying architectural frameworks that facilitate these transitions. Figure 4 illustrates the hierarchical structure of Google Spanner and NewSQL technologies, detailing the architectural components and features of Google Spanner. This figure not only emphasizes the transition from NoSQL to NewSQL but also elucidates the characteristics and applications of NewSQL databases. Furthermore, it provides a comparative analysis of NewSQL, NoSQL, and SQL databases, thereby highlighting current applications and future research directions in the field. Such visual representation enhances our comprehension of the intricate relationships and functionalities within these database paradigms, allowing for a more informed discussion on their implications in contemporary data management practices.

4 Google Spanner and NewSQL

4.1 Overview of Google Spanner

Google Spanner revolutionizes distributed database design by integrating the strong consistency of relational databases with NoSQL's horizontal scalability. Central to its architecture is the TrueTime API, providing a globally synchronized clock that ensures external consistency and real-time transaction ordering across geographically distributed nodes, crucial for applications demanding stringent consistency over vast distances [21]. Spanner's tiered architecture, akin to Handoff Counters, efficiently manages state transitions and value handoffs between nodes without a centralized sequencer, enhancing scalability and consistency [51]. It also utilizes a fragmented object approach similar to COBFS, breaking large datasets into smaller blocks to boost concurrency and scalability [52].

This is further illustrated in Figure 5, which depicts the hierarchical structure of Google Spanner's key components, focusing on architecture, performance, and security aspects. The figure highlights the integration of the TrueTime API, tiered architecture, and fragmented objects in its design, as well as the adaptive learning mechanisms and multi-data center deployment that enhance performance. Additionally, it underscores the use of authenticated data structures for security and consistency.

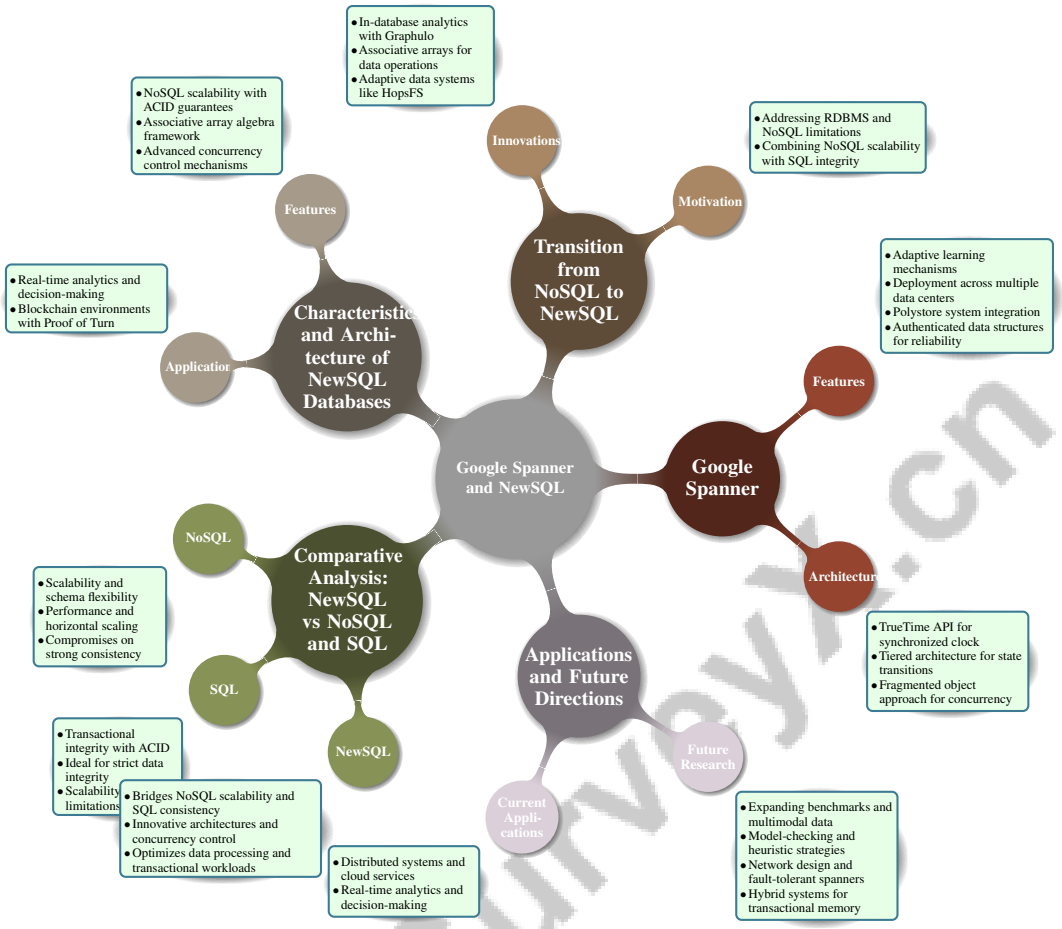


Figure 4: This figure illustrates the hierarchical structure of Google Spanner and NewSQL technologies, detailing the architectural components and features of Google Spanner, the transition from NoSQL to NewSQL, the characteristics and applications of NewSQL databases, and a comparative analysis of NewSQL, NoSQL, and SQL databases. It also highlights current applications and future research directions in the field.

Spanner's adaptive learning mechanisms enable it to evolve based on operational feedback, maintaining performance and reliability in dynamic, large-scale environments [53]. Its deployment across multiple data centers, as demonstrated by the LEGOStore prototype on Google Cloud Platform, optimizes cost and latency while maintaining a linearizable data store [21]. Additionally, Spanner embodies the polystore system concept, integrating diverse data management paradigms for versatile data integration strategies [9]. The use of authenticated data structures in protocols like VICOS ensures a consistent operational view across multiple clients, enhancing cloud object storage systems' reliability [2].

4.2 Transition from NoSQL to NewSQL

The transition from NoSQL to NewSQL databases marks a significant evolution in database systems, addressing the limitations of RDBMS and NoSQL in managing OLTP workloads amid big data challenges [17]. While NoSQL databases address traditional RDBMS scalability issues with flexible schemas and horizontal scaling, they often compromise on ACID properties, limiting their applicability in scenarios requiring strong transactional consistency [7]. NewSQL databases bridge this gap by combining NoSQL scalability with SQL's transactional integrity, meeting the demand for efficient data management systems capable of handling high-velocity data while maintaining consistency and supporting complex queries [17].

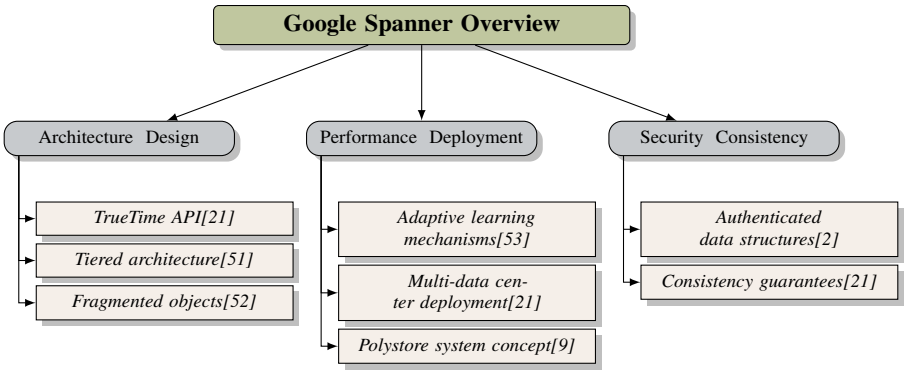


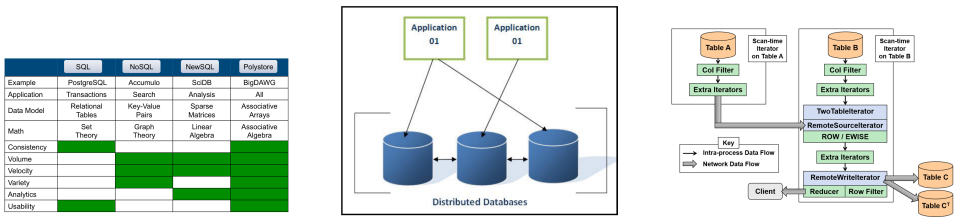
Figure 5: This figure illustrates the hierarchical structure of Google Spanner's key components, focusing on architecture, performance, and security aspects. It highlights the integration of TrueTime API, tiered architecture, and fragmented objects in its design, adaptive learning and multi-data center deployment for performance, and the use of authenticated data structures for security and consistency.

The integration of in-database analytics, as seen in Apache Accumulo through Graphulo, exemplifies enhanced data processing capabilities [54]. Associative arrays serve as a unified mathematical framework, optimizing operations across different database types and simplifying interactions among SQL, NoSQL, and NewSQL databases [10]. The D4M software system demonstrates the potential of associative arrays for enhancing data integration and processing efficiency [55]. NewSQL's evolution includes adaptive data systems that dynamically adjust to data and queries, contrasting with the static nature of traditional systems. Implementations like HopsFS, which replace the single-node HDFS approach with a scalable NewSQL-based solution, illustrate NewSQL's advantages in efficient metadata management and scalability [33].

4.3 Characteristics and Architecture of NewSQL Databases

NewSQL databases are designed to deliver NoSQL scalability while preserving the ACID guarantees of traditional SQL databases, excelling in high-velocity data processing and complex transactional workloads. A defining feature is their reliance on associative array algebra, which provides a unified framework for operations across SQL, NoSQL, and NewSQL systems [32]. Advanced concurrency control mechanisms, such as Lazy State Determination (LSD), enhance performance by minimizing the overhead of maintaining strong consistency across distributed nodes, indicating potential directions for future research [56].

NewSQL systems improve join performance and query expressiveness, particularly in materialized views, efficiently managing complex queries and transformations [57]. This capability is crucial for applications requiring real-time analytics and data-driven decision-making. Integrating qualitative and quantitative analyses into NewSQL methodologies enhances operational efficiency, as demonstrated by applications in blockchain environments. For instance, the Proof of Turn (PoT) mechanism reduces operational costs for game publishers while fostering player trust and engagement, highlighting NewSQL architectures' versatility across diverse use cases [11].



(a) NoSQL Database Systems Comparison[10]

(b) Distributed Databases[17]

(c) Scalable MapReduce Framework[54]

Figure 6: Examples of Characteristics and Architecture of NewSQL Databases

As depicted in Figure 6, NewSQL databases epitomize a pivotal evolution in database management, particularly in addressing traditional SQL and NoSQL systems' constraints. Google Spanner exemplifies this architecture, offering a unique combination of scalability and consistency. NewSQL databases maintain ACID properties while providing the horizontal scalability characteristic of NoSQL systems. Their architecture integrates distributed systems, scalable frameworks like MapReduce, and innovative data models tailored for high-volume, high-velocity data, emphasizing real-time analytics and usability [10, 17, 54].

4.4 Comparative Analysis: NewSQL vs NoSQL and SQL

Benchmark	Size	Domain	Task Format	Metric
BPC[42]	1,000,000	Blockchain	Consensus Evaluation	Throughput, Latency
CRB[58]	10,000	Logical Reasoning	Multi-step Reasoning	Accuracy, F1-score

Table 3: This table presents a selection of representative benchmarks used for evaluating performance in various domains, including blockchain and logical reasoning. It details the size of each benchmark, the specific domain of application, the task format, and the metrics employed for assessment, providing a comprehensive overview of their characteristics.

A comparative analysis of NewSQL, NoSQL, and traditional SQL databases reveals their distinct strengths and weaknesses, reflecting their design philosophies and intended applications. Table 3 provides a detailed overview of representative benchmarks relevant to the comparative analysis of database systems, highlighting their application domains, task formats, and evaluation metrics. NoSQL systems emerged to overcome SQL databases' limitations, particularly regarding scalability and schema flexibility, prioritizing performance and horizontal scaling. However, this often compromises the strong consistency and security guarantees provided by traditional RDBMS [17]. Traditional SQL databases ensure robust transactional integrity through ACID properties, ideal for applications demanding strict data integrity. However, their scalability limitations can hinder modern applications requiring high throughput and low latency in distributed environments.

NewSQL databases bridge the gap between NoSQL scalability and SQL consistency guarantees, leveraging innovative architectures and concurrency control mechanisms to deliver high performance while maintaining ACID compliance. This balance of performance, security, and consistency positions NewSQL as an attractive option for applications requiring both NoSQL scalability and traditional RDBMS reliability. By integrating associative array algebra and advanced data management techniques, NewSQL systems optimize data processing and efficiently support complex transactional workloads [17].

4.5 Applications and Future Directions

NewSQL databases have gained prominence in distributed systems, providing robust solutions for applications necessitating high scalability and strong consistency, particularly in OLTP, real-time analytics, and cloud services. By utilizing associative array algebra, NewSQL databases enhance data processing and facilitate real-time decision-making, essential in data-intensive environments [32]. In cloud computing, NewSQL databases offer scalable and reliable infrastructure for distributed applications, ensuring data consistency and availability across geographically dispersed data centers, crucial for maintaining seamless user experiences and supporting global operations [59].

Looking ahead, research on NewSQL databases is set to explore promising avenues, including expanding benchmarks to encompass additional reasoning tasks and integrating multimodal data to improve evaluation methods [58]. Applying model-checking-driven explorative approaches to a broader range of distributed systems and employing heuristic strategies could optimize the model-checking process [60]. Advancements in network design and distributed systems, such as developing fault-tolerant spanners, offer insights into enhancing NewSQL architectures' robustness and scalability [61]. Investigating approximately universal optimality in weighted graphs and other spanner classes may further improve network design and graph algorithms, indicating potential applications for optimizing NewSQL databases [62]. Exploring hybrid systems that support both locking and non-blocking structures could yield comprehensive solutions for transactional memory, addressing current limitations and expanding NewSQL databases' applicability [5].

5 Sharding and Scalability

Scalability is crucial in distributed systems, influencing data management efficacy and performance. Sharding, the division of data into smaller, manageable segments, is pivotal for achieving scalability. It facilitates workload distribution across multiple nodes, enhancing fault tolerance and resource utilization. The following subsection explores the principles of sharding in distributed systems.

5.1 Concept of Sharding in Distributed Systems

Sharding is essential for scalability in distributed systems, allowing databases to manage increased loads by distributing data across nodes, optimizing resource use, and improving performance. It is linked to lightweight bounded-degree spanners, which enhance communication and computation efficiency [63]. By dividing databases into independently managed shards, sharding supports horizontal scaling, accommodating large data volumes and high transaction rates without performance loss. This distribution improves load balancing and fault tolerance, ensuring system resilience to shard failures [33]. Dependable spanners, which maintain connectivity despite edge failures, further bolster the reliability and scalability of sharded systems [27].

Advanced sharding techniques align with evolutionary data systems, using modular architectures for flexible storage transitions and scalability enhancement. Materialized views (MVs) precompute expensive joins, improving performance in NoSQL databases and demonstrating sharding's versatility in optimizing query execution [57]. Polyglot data management approaches, which partition databases for handling multiple data models, highlight sharding's role in distributed environments.

Sharding's effectiveness in distributed databases is linked to constructing spanners for metric spaces that maintain properties despite vertex failures, enhancing resilience [64]. Utilizing snapshots to maintain operation order simplifies implementation and boosts scalability.

Sharding partitions large datasets into manageable fragments, facilitating independent manipulation and improved concurrency. This approach enhances performance while maintaining strong consistency guarantees, as demonstrated by systems like COBFS, which use fragmented objects to boost concurrent access to large shared data. Advances in distributed metadata management, such as HopsFS, show how effective sharding alleviates bottlenecks, enabling larger clusters and higher throughput compared to traditional systems like HDFS. Sharding is vital for optimizing data management and access in distributed environments, aligning with evolving definitions of consistency and availability in the context of the CAP theorem [65, 16, 33, 52, 19]. By leveraging spanner construction and fault tolerance concepts, sharding allows distributed databases to efficiently handle increased loads and data volumes, ensuring robust performance and reliability in modern computing environments.

5.2 Spanner Construction and Its Relation to Sharding

The construction of Google Spanner, particularly its sharding implementation, is crucial for enhancing scalability and reliability in distributed databases, managing large-scale data across multiple nodes while maintaining consistent performance and fault tolerance [66, 67]. Google Spanner employs advanced spanner construction techniques to manage vast data amounts across geographically distributed nodes, ensuring high availability and consistency despite network partitions and node failures.

A key aspect of Google Spanner's architecture is its hierarchical partitioning methods, related to constructing dependable spanners that can endure high edge failure rates while minimizing vertex pair disconnections [27]. This method enhances fault tolerance by preserving connectivity and data integrity despite node or network path failures, essential for Google Spanner's reliability.

Google Spanner also utilizes distributed algorithms for spanner construction, such as the Distributed-Spanner algorithm, which relies on localized information for efficient construction [63]. These algorithms optimize communication and coordination among nodes, reducing overhead in maintaining a consistent state across the distributed system. Leveraging such algorithms, Google Spanner enhances data replication and synchronization processes, contributing to its scalability.

The system architecture incorporates techniques similar to Synergy, combining schema-based and workload-driven materialized views with lightweight concurrency control to optimize data manage-

ment in NoSQL databases [57]. This method allows efficient data management and query execution, bolstering the system's scalability by optimizing communication and computation across distributed nodes.

Moreover, Google Spanner employs sparse spanner construction methods, such as incrementally adding edges based on local connectivity, ensuring efficient and independent operation of each shard [64]. This approach enhances the system's ability to manage increased loads and data volumes, making it a scalable solution for modern distributed applications.

Evaluating reliable spanners across various metric spaces, including trees and planar graphs, demonstrates improvements in maintaining connectivity and performance, further highlighting Spanner's architecture in optimizing performance [64]. Organizing net points into incubators for managing connections and maintaining low degree also contributes to the robustness and efficiency of spanner construction [68].

5.3 Reliable Spanners and Fault Tolerance

Reliable spanners are vital for the fault tolerance of distributed systems, ensuring connectivity and performance during node or edge failures. These spanners are constructed using sophisticated algorithms that balance efficiency with resilience, enabling seamless operations despite disruptions [69].

A significant challenge in constructing reliable spanners is balancing the spanner's stretch and maximum degree. Many existing methods struggle to limit both effectively. However, recent advancements have introduced improved deterministic distributed spanner constructions that significantly reduce communication rounds and message sizes, enhancing construction efficiency [70].

Resilient graph spanners have improved local resilience to edge failures and better maintain distance properties compared to traditional fault-tolerant spanners [71]. This enhancement is vital for distributed systems, where maintaining connectivity and performance amidst failures is essential. Constructing fault-tolerant spanners that preserve distance properties during vertex failures ensures reliable operations in distributed systems, providing a framework for maintaining connectivity despite disruptions [27].

Additionally, reliable spanners that require fewer edges than previous constructions are practical for real-world applications [68]. These spanners maintain robustness while reducing edge count, making them more space-efficient. The proposed method achieves spanner sizes comparable to undirected graphs, a significant advantage previously unattainable for directed graphs [64].

Reliable spanners also benefit from improved round complexity, allowing faster construction without compromising edge count [64]. This improvement is crucial for distributed systems, as minimizing spanner construction time can significantly enhance overall system performance. The proposed reliable spanners are designed to maintain fault tolerance, ensuring distributed systems remain resilient and efficient in dynamic environments.

5.4 Advanced Techniques for Scalability

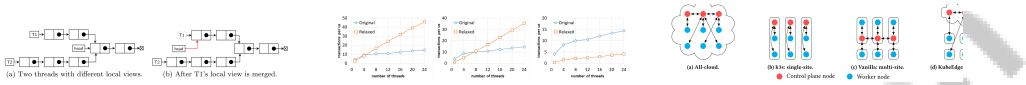
Enhancing scalability in distributed systems requires innovative techniques beyond traditional sharding. One advancement is the development of reliable spanners, crucial for maintaining performance and connectivity amidst significant node failures. Constructing these spanners involves two steps: first, constructing spanners for uniform metrics and then using these constructions to address general metrics through covers [64]. This method introduces a novel construction approach that transforms spanners for uniform metrics into reliable spanners for general metrics, significantly improving performance and scalability [64].

Dynamic partitioning strategies also present a promising avenue for scalability enhancement, particularly in systems like HopsFS, where metadata operation hotspots present challenges. Future research should explore these strategies to improve performance and address scalability issues associated with concentrated data access patterns [33].

Integrating locality-sensitive orderings in reliable spanner construction enhances scalability by improving stretch and sparsity while withstanding massive node failures. This approach extends to broader metric classes, offering robust solutions for distributed systems facing significant disruptions

[64]. The development of relaxed spanners for directed disk graphs suggests potential advancements in compact routing schemes and distance oracles, leveraging insights from this work to improve scalability in distributed systems.

These advanced techniques significantly enhance distributed systems' scalability by optimizing communication, computation, and data management, while addressing the trade-offs between consistency and availability as highlighted by the CAL theorem, which quantifies the relationship between latency, inconsistency, and unavailability. Employing both centralized and decentralized coordination mechanisms enables systems to maintain bounded inconsistency or unavailability under network partitions, thereby enhancing overall performance and reliability [16, 18]. By moving beyond traditional sharding methods, these approaches ensure that distributed systems can efficiently handle increased loads and complex data structures, maintaining robust performance in dynamic environments.



(a) Two threads with different local views[72]

(b) Performance Comparison of Original and Relaxed Algorithms in Distributed Systems[73]

(c) Understanding Cloud Computing: Different Architectures and Their Applications[34]

Figure 7: Examples of Advanced Techniques for Scalability

As shown in Figure 7, scalability remains a critical concern in distributed systems and cloud computing as the demand for efficient, high-performance systems continues to grow. The discussed examples focus on advanced techniques for scalability, emphasizing sharding and other strategies to enhance system performance. The first illustration, "Two threads with different local views," highlights how threads interact with shared memory structures, underscoring the importance of understanding memory access patterns for optimizing thread performance. The second image, "Performance Comparison of Original and Relaxed Algorithms in Distributed Systems," examines trade-offs between original and relaxed algorithms, showcasing their respective efficiencies in managing increased thread counts and transaction loads. Finally, the third image, "Understanding Cloud Computing: Different Architectures and Their Applications," provides a comparative analysis of various cloud architectures, elucidating how different network topologies and resource allocations can impact scalability and performance. Together, these examples underscore the significance of employing advanced techniques to tackle the challenges of scalability in modern distributed systems [72, 73, 34].

6 Consistency Models: Linearizability and Eventual Consistency

In distributed systems, consistency models are pivotal for ensuring the correctness and reliability of concurrent operations. This section analyzes two key models: linearizability and eventual consistency. Linearizability ensures operation order and atomicity, whereas eventual consistency emphasizes high availability and partition tolerance. A comprehensive understanding of these models is crucial for system designers to balance strict consistency with operational efficiency. The following subsection delves into linearizability, elucidating its principles and significance in concurrent data structures.

6.1 Understanding Linearizability

Linearizability is a core consistency model in distributed systems, ensuring operations on shared objects appear instantaneous between invocation and completion, serving as a correctness criterion for concurrent data types [74]. It allows reasoning about concurrent data structures as if operations were executed sequentially, respecting real-time ordering [4]. This model is vital for verifying the correctness of concurrent algorithms, ensuring behavior aligns with sequential specifications. Interval-linearizability offers a refined representation of concurrent behaviors, enhancing the understanding of operation interactions [4].

Despite its advantages, linearizability can be computationally complex, especially in operations involving intricate interactions within concurrent data structures. Efficient verification methods are necessary to manage this complexity [74]. Linearizability ensures concurrent operations appear

atomic and sequential, providing essential correctness guarantees. It requires preserving the real-time order of non-overlapping operations while allowing overlapping operations in any order. This model is instrumental in verifying concurrent data structures and is widely applied in state-machine replication for fault tolerance. However, it has limitations in handling scenarios like conditional waits and nested invocations. Recent discussions highlight the need for alternative models, such as interval linearizability, which relaxes constraints of traditional linearizability, broadening replication techniques' applicability in distributed systems [75, 76, 77].

6.2 Challenges in Proving Linearizability

Proving linearizability in complex distributed environments is challenging due to the intricate nature of concurrent algorithm implementations and the dynamic identification of linearization points (LPs). The dynamic nature of LPs, often dependent on runtime conditions, complicates verification, especially for algorithms with future-dependent linearizations. This complexity is further exacerbated by the entanglement of concurrency and persistency aspects in non-volatile memory (NVM) semantics, which existing methods often inadequately address [78].

The computational complexity involved in proving linearizability is significant, as the problem is NP-complete, requiring exploration of a vast search space of possible executions [74]. This complexity intensifies in systems utilizing randomized algorithms, where execution path unpredictability complicates proofs [79]. The CLAM theorem highlights fundamental limitations in simultaneously satisfying certain consistency properties in distributed systems, complicating the pursuit of linearizability [80].

In fully asynchronous systems, the primary challenge is the impossibility of determining the real-time order of events, crucial for establishing linearizability [81]. This challenge is compounded by strict synchronization requirements imposed by linearizability, which can hinder performance and scalability [82]. Verifying specific operations, such as the Contains operation in lazy set objects, requires advanced techniques due to its nature of allowing operations to set linearization points for others [83].

Addressing these challenges necessitates ongoing research and innovation to enhance both theoretical foundations and practical applications. Various techniques for verifying linearizability have emerged, each with distinct formal foundations such as refinement, shape analysis, and reduction. However, differences in frameworks and terminology can complicate practitioners' ability to select suitable methods for specific data structures. Recent advancements, including proof methods accommodating future-dependent linearizations and generic logics for concurrency reasoning, highlight the necessity of evolving verification techniques to ensure comprehensive adaptability to challenges posed by concurrent data structures and state-machine replication, where traditional linearizability may prove too restrictive [84, 85, 76, 77].

6.3 Eventual Consistency and Its Variants

Eventual consistency is a key consistency model in distributed systems, characterized by the guarantee that updates to a data store will eventually propagate to all replicas, allowing the system to converge to a consistent state over time. This model is particularly beneficial in scenarios prioritizing high availability and partition tolerance over immediate consistency, making it suitable for applications with high read and write throughput requirements. Almeida et al. expand the range of consistency models to include those capable of handling synchronization-oriented abstractions and cyclic visibility, essential for understanding the nuances of eventual consistency [80].

The epistemic perspective on consistency further elucidates eventual consistency by framing it as a special case of sequential consistency, where the system's knowledge about the order of operations is gradually refined as updates propagate. This perspective aids in comprehending the trade-offs involved, particularly in environments where immediate consistency is not strictly necessary. However, eventual consistency presents challenges for reliable distributed applications, primarily due to the lack of immediate visibility of updates across replicas, leading to scenarios where different replicas may reflect outdated or inconsistent states for an indeterminate period. Moreover, eventual consistency requires complex and often non-intuitive distributed specification techniques to manage the convergence of replicas, complicating implementation and understanding. Consequently, while eventual consistency allows for high availability and fault tolerance, it introduces complexities that can

undermine the reliability of distributed applications [65, 29]. This limitation necessitates additional mechanisms to ensure applications can tolerate temporary inconsistencies without compromising correctness.

Variants of eventual consistency, such as causal consistency, enhance the consistency model by maintaining the causal relationships between operations. This ensures that causally related updates are observed in the correct order, crucial for applications dependent on the integrity of these relationships. Causal consistency allows for greater availability and partition tolerance compared to stronger consistency models while providing a framework for verifying that executions conform to these causal relationships. By leveraging tools that can automatically check for causal consistency, developers can ensure their distributed systems operate correctly even amid network partitions or failures [86, 87, 26, 88]. This variant is particularly useful in collaborative applications where the order of operations impacts the final outcome. Furthermore, staleness-based models refine eventual consistency by bounding the time or versions by which replicas may lag behind, offering a balance between consistency and performance.

6.4 Trade-offs Between Linearizability and Eventual Consistency

The trade-offs between linearizability and eventual consistency are central to designing distributed systems, directly influencing performance and reliability. Linearizability, a strong consistency model, ensures that operations appear atomic and are executed instantaneously, maintaining real-time order across distributed nodes. This model is indispensable for applications demanding strict data integrity, such as financial systems, where deviations from the expected order can lead to critical errors [80]. However, achieving linearizability involves significant synchronization overhead, often resulting in increased latency and reduced system throughput. The complexity of verifying linearizability, underscored by the EXPSPACE-hard classification for certain operations, further highlights the computational challenges associated with maintaining strict consistency [74].

Eventual consistency offers a more relaxed model that enhances system performance by allowing asynchronous updates across replicas. This model prioritizes high availability and partition tolerance, permitting temporary inconsistencies that eventually resolve. The flexibility of eventual consistency makes it suitable for applications where immediate consistency is not critical, allowing for improved scalability and reduced coordination overhead. However, many commonly held beliefs about strong versus eventual consistency are oversimplified and misleading, indicating the complexity of these trade-offs [18]. The CALM Theorem provides a positive framework for understanding consistency in distributed systems, delineating which programs can be executed without coordination [1].

A limitation of eventual consistency is the challenge of ensuring data consistency across different systems, leading to potential gaps in data availability and integrity. The trade-offs between relying on a trusted server and using protocols like VICOS for integrity and consistency verification illustrate the challenges of maintaining reliability in distributed systems [2]. Additionally, the Pathway Theorem introduces a structured methodology for proving durable linearizability, allowing for a modular approach to verification, crucial for addressing these challenges [78].

Innovative approaches, such as Intermediate Value Linearizability (IVL), relax linearizability to allow for more cost-effective implementations while preserving probabilistic error bounds. These approaches highlight ongoing efforts to balance the stringent requirements of linearizability with the flexibility of eventual consistency [89]. The analysis of trade-offs between linearizability and write strong-linearizability further emphasizes the importance of preserving termination properties in randomized algorithms [79].

Ultimately, the decision between linearizability and eventual consistency should be guided by the specific application requirements, considering factors such as the need for immediate consistency, tolerance for temporary inconsistencies, and the desired balance between performance and reliability. By understanding these trade-offs, system designers can make informed choices that align with their application's goals, ensuring robust and efficient operations in distributed environments [4].

6.5 Innovative Approaches and Future Directions

Innovative approaches to consistency models in distributed systems are continuously evolving to address the complex demands of modern computing environments. One promising direction involves

refining locality-sensitive orderings, enhancing their applicability across a broader range of metric spaces, potentially leading to advancements in computational geometry and graph theory [69]. This refinement could improve distributed system performance by optimizing the construction of reliable spanners, which are crucial for maintaining connectivity and performance even amidst significant node failures [71].

The development of vertex fault-tolerant (VFT) k-spanners, achieving fault tolerance with only a sublinear increase in spanner size, represents a significant improvement over previous approaches [90]. Future research could explore applying these spanners in various distributed scenarios, enhancing resilience and efficiency. Additionally, further refinements to local linearizability could enhance its applicability in complex data structures and its integration with other consistency models, offering more robust solutions for concurrent programming scenarios [82].

In linearizability, the Abstract History Method provides a systematic approach to proving linearizability through abstraction, simplifying the verification process for complex distributed algorithms [83]. Future research will focus on extending this method to other complex data structures and exploring its potential for automation in proving linearizability [85]. Additionally, exploring deterministic specifications and broader classes of specifications could yield new insights into the complexity of linearizability [91].

Optimistic execution techniques present another avenue for innovation, integrating monitoring approaches with automated rollback systems and developing more efficient predicate detection algorithms [24]. These advancements could lead to more efficient verification processes and robust consistency guarantees in distributed systems.

Furthermore, applying the Pathway Theorem to other types of persistent data structures could optimize flushing strategies and enhance the durability of linearizability in distributed systems [78]. By exploring these innovative approaches and future directions, researchers can enhance the reliability and efficiency of distributed systems, providing robust solutions to modern computing challenges.

Future work could focus on refining existing frameworks to better handle specific types of distributed systems and exploring additional properties that could enhance consistency guarantees [80]. This exploration is crucial for developing more adaptable and robust consistency models that meet the diverse needs of contemporary distributed applications.

7 Distributed Consensus and Scalability Challenges

7.1 Challenges in Achieving Distributed Consensus

Achieving distributed consensus in large-scale systems presents significant challenges, primarily concerning consistency, coordination, and fault tolerance in dynamic environments. A critical issue is maintaining linearizability across geographically distributed data centers while minimizing latency and operational costs. This is particularly complex during dynamic workloads and potential data center failures, necessitating robust synchronization mechanisms to maintain a consistent state across distributed nodes [21]. The verification process is further complicated by the exponential growth of the search space for checking linearizability, which can be as large as O(N!) for a history of length N [74].

Integrating new data stores often disrupts seamless query processing across heterogeneous systems, requiring consensus mechanisms that accommodate diverse environments while ensuring efficient data management [9]. Verifying linearizability in concurrent data structures, such as the Lazy Set algorithm, demands a mathematical framework capable of proving correctness amidst concurrent operations [83].

In environments with potentially malicious servers, ensuring data integrity and consistency is crucial. Protocols like VICOS enable clients to verify data integrity and consistency without fully trusting the server [2]. Existing methods often struggle with concurrent operations without incurring high implementation costs, posing a significant obstacle to efficient distributed consensus [89].

The presence of strong adversaries capable of manipulating execution schedules to prevent termination underscores the need for robust consensus mechanisms that withstand adversarial conditions [79]. Traditional coordination mechanisms can lead to performance bottlenecks, prompting exploration of problems solvable without coordination to enhance system efficiency [1].

The construction of spanners with non-constant lightness bounds limits their practical applicability in distributed settings, emphasizing the need for more efficient spanner construction techniques to support consensus processes [63]. Addressing these challenges requires ongoing research and innovation to develop robust and scalable distributed consensus mechanisms capable of managing the complexities of large-scale, dynamic environments.

7.2 Innovative Algorithms and Techniques for Consensus

Innovative algorithms and techniques are essential for advancing consensus processes in distributed systems, addressing challenges such as coordination, fault tolerance, and communication efficiency. The Distributed-Spanner algorithm exemplifies this advancement by using localized information to optimize communication and coordination among nodes, significantly reducing communication overhead while maintaining efficiency as systems scale [63].

Intermediate Value Linearizability (IVL) represents a significant improvement over traditional linearizability by permitting intermediate return values, facilitating cost-effective implementations, and enhancing consensus mechanisms' overall efficiency [89]. This flexible framework balances consistency and performance, making it suitable for various distributed applications.

The P-compositionality based linearizability checker (PCL) partitions operation histories into independent subproblems, allowing for parallel verification of linearizability and reducing computational complexity [74]. By breaking down complex operations into manageable components, PCL enhances the efficiency of consensus processes.

Developments in write strong-linearizability ensure that write operations are strongly linearizable, providing robust guarantees for data consistency and reliability [79]. This focus on the linearizability of write operations addresses critical challenges in maintaining consistency across distributed nodes.

The Lazy Set algorithm's two-layer proof structure offers a higher abstraction level for defining axioms and a lower level for analyzing a simplified version of the algorithm [83]. This structured framework enhances the reliability of consensus mechanisms by verifying the correctness of concurrent data structures.

These innovative algorithms and techniques signify substantial progress in distributed consensus, particularly for permissioned blockchains and Directed Acyclic Graph (DAG)-based protocols. They provide robust solutions for achieving agreement in complex environments, addressing issues such as security, performance, and consistency. For example, Proof-of-Authority protocols emphasize reliable consistency guarantees for data integrity, while the Nezha protocol demonstrates enhanced throughput and reduced latency through high-accuracy clock synchronization. Collectively, these contributions improve the resilience and efficiency of consensus mechanisms and pave the way for future research to tackle existing challenges and optimize performance in real-world applications [43, 39, 42, 38]. By leveraging advanced consistency models, localized information, and structured verification frameworks, these approaches enhance the reliability and efficiency of consensus mechanisms, supporting the development of scalable and resilient distributed systems.

7.3 Strategies for Overcoming Scalability Challenges

Overcoming scalability challenges in distributed consensus mechanisms necessitates a multifaceted approach, incorporating innovative strategies and best practices to enhance performance and efficiency. One promising strategy involves developing improved deterministic distributed construction methods for graph spanners, optimizing communication efficiency and reducing overhead associated with maintaining consensus across distributed nodes. Future research should extend these methods to accommodate diverse graph structures and optimize algorithms for specific applications, thereby enhancing scalability [92].

Integrating advanced consensus protocols, such as DAG-based protocols, presents another avenue for addressing scalability challenges. These protocols enhance transaction throughput and reduce latency by enabling parallel processing of transactions, which is essential for maintaining performance in large-scale systems. Future research should focus on developing standardized performance metrics, exploring fairness in transaction handling, and enhancing privacy mechanisms to ensure robust and scalable consensus solutions [43].

Deploying high-performance consensus mechanisms like Nezha underscores the importance of optimizing clock synchronization techniques and integrating advanced cloud technologies to enhance performance and reliability. Refining these aspects allows distributed systems to achieve greater scalability, accommodating increased loads and complex operations without compromising efficiency [38].

Exploring optimizations for communication efficiency in message-passing models can significantly contribute to overcoming scalability challenges. Investigating alternative spanners and refining existing communication protocols can reduce latency and improve coordination among nodes, thereby enhancing overall scalability [93].

8 Conclusion

The survey of distributed systems and databases underscores the pivotal role of several foundational concepts and technologies in constructing resilient and scalable infrastructures. The examination of the CAP theorem, ACID and BASE models, consensus algorithms, and consistency frameworks reveals their substantial impact on the reliability and efficiency of distributed architectures. The CAP theorem elucidates the necessary trade-offs between consistency, availability, and partition tolerance, guiding system architects in making informed design decisions. The ACID and BASE paradigms further shape the equilibrium between consistency and availability in database transaction management.

Consensus algorithms, particularly Paxos and Raft, are integral to achieving consensus across distributed nodes, thereby ensuring data integrity and system robustness. Innovations such as the Egalitarian Paxos (EPaxos) protocol have demonstrated enhancements in scalability and effectiveness, despite certain constraints. The transition from NoSQL to NewSQL databases, exemplified by Google Spanner, signifies progress in meeting the scalability and consistency demands of contemporary data-intensive applications.

Sharding and scalability techniques remain essential in navigating the complexities of distributed systems. Methods like hierarchical partitions and spanner constructions improve system performance and fault tolerance, with lightweight bounded-degree spanners offering insights into optimizing network design. Consistency models, such as linearizability and eventual consistency, provide tailored levels of data integrity and availability for specific applications, with the classification of eventually linearizable objects offering deeper understanding of their synchronization capabilities.

In light of the current state of distributed systems and databases, continuous research and innovation are imperative to address emerging challenges and enhance performance. Future endeavors should focus on developing programming languages and tools that facilitate the creation of monotonic distributed systems, as well as exploring the expressive potential of monotonic programs. Additionally, advancements in polyglot data management frameworks are crucial for adapting to evolving data requirements and technological shifts. Mastery of these core concepts is essential for building distributed systems that are robust, scalable, and adaptable to the dynamic demands of modern computing environments. As the field progresses, the integration of novel algorithms and techniques will be crucial in shaping the future of distributed systems and databases, ensuring their continued relevance and efficacy in addressing complex data management challenges.

References

[1] Joseph M. Hellerstein and Peter Alvaro. Keeping calm: When distributed consistency is easy, 2019.

[2] Marcus Brandenburger, Christian Cachin, and Nikola Knežević. Don't trust the cloud, verify: Integrity and consistency for cloud object stores, 2016.

[3] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain cap theorem allows user-dependent adaptivity and finality, 2021.

[4] Armando Castaneda, Michel Raynal, and Sergio Rajsbaum. Specifying concurrent problems: Beyond linearizability, 2015.

[5] Wentao Cai, Haosen Wen, Vladimir Maksimovski, Mingzhe Du, Rafaello Sanna, Shreif Abdallah, and Michael L. Scott. Fast nonblocking persistence for concurrent data structures, 2021.

[6] Pai Zeng, Zhenyu Ning, Jieru Zhao, Weihao Cui, Mengwei Xu, Liwei Guo, Xusheng Chen, and Yizhou Shan. The cap principle for llm serving: A survey of long-context large language model serving, 2024.

[7] Mayank Raikwar, Danilo Gligoroski, and Goran Velinov. Trends in development of databases and blockchain, 2020.

[8] Stratos Idreos, Lukas M. Maas, and Mike S. Kester. Evolutionary data systems, 2017.

[9] Daniel Glake, Felix Kiehn, Mareike Schmidt, Fabian Panse, and Norbert Ritter. Towards polyglot data stores – overview and open research questions, 2022.

[10] Hayden Jananthan, Ziqi Zhou, Vijay Gadepally, Dylan Hutchison, Suna Kim, and Jeremy Kepner. Polystore mathematics of relational algebra, 2017.

[11] Dominik Braun. Proof-of-turn: Blockchain consensus using a round-robin procedure as one possible solution for cutting costs in mobile games, 2023.

[12] Hagit Attiya, Armando Castañeda, and Constantin Enea. Strong linearizability using primitives with consensus number 2, 2024.

[13] Christopher S. Meiklejohn. On the design of distributed programming models, 2017.

[14] Christian Cachin and Olga Ohrimenko. Verifying the consistency of remote untrusted services with conflict-free operations, 2018.

[15] Florian Jacob, Carolin Beer, Norbert Henze, and Hannes Hartenstein. Analysis of the matrix event graph replicated data type, 2020.

[16] Edward A. Lee, Soroush Bateni, Shaokai Lin, Marten Lohstroh, and Christian Menard. Quantifying and generalizing the cap theorem, 2021.

[17] A B M Moniruzzaman. Newsql: Towards next-generation scalable rdbms for online transaction processing (oltp) for big data management, 2014.

[18] Martin Kleppmann. A critique of the cap theorem, 2015.

[19] Priyam Shah, Jie Ye, and Xian-He Sun. Survey the storage systems used in hpc and bda ecosystems, 2021.

[20] Mohammed S. Al-Mahfoudh, Ryan Stutsman, and Ganesh Gopalakrishnan. Efficient linearizability checking for actor-based systems, 2023.

[21] Hamidreza Zare, Viveck R. Cadambe, Bhuvan Urgaonkar, Chetan Sharma, Praneet Soni, Nader Alfares, and Arif Merchant. Legostore: A linearizable geo-distributed store combining replication and erasure coding, 2022.

[22] A B M Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison, 2013.

[23] Eleni Bila, Simon Doherty, Brijesh Dongol, John Derrick, Gerhard Schellhorn, and Heike Wehrheim. Defining and verifying durable opacity: Correctness for persistent software transactional memory, 2020.

[24] Duong Nguyen, Aleksey Charapko, Sandeep Kulkarni, and Murat Demirbas. Technical report: Optimistic execution in key-value store, 2018.

[25] Sandeep Kulkarni, Duong Nguyen, Lewis Tseng, and Nitin Vaidya. Impact of the consistency model on checkpointing of distributed shared memory, 2022.

[26] Rachid Zennou, Ranadeep Biswas, Ahmed Bouajjani, Constantin Enea, and Mohammed Erradi. Checking causal consistency of distributed databases, 2021.

[27] Sariel Har-Peled and Maria C. Lusardi. Dependable spanners via unreliable edges, 2024.

[28] Kartik Nagar, Prasita Mukherjee, and Suresh Jagannathan. Semantics, specification, and bounded verification of concurrent libraries in replicated systems, 2020.

[29] Matthieu Perrin, Achour Mostéfaoui, and Claude Jard. Brief announcement: Update consistency in partitionable systems, 2015.

[30] Edward A. Lee, Ravi Akella, Soroush Bateni, Shaokai Lin, Marten Lohstroh, and Christian Menard. Consistency vs. availability in distributed real-time systems, 2023.

[31] Marc Shapiro, Annette Bieniusa, Nuno Preguiça, Valter Balegas, and Christopher Meiklejohn. Just-right consistency: reconciling availability and safety, 2018.

[32] Jeremy Kepner, Vijay Gadepally, Dylan Hutchison, Hayden Jananthan, Timothy Mattson, Siddharth Samsi, and Albert Reuther. Associative array model of sql, nosql, and newsql databases, 2016.

[33] Salman Niazi, Mahmoud Ismail, Steffen Grohsschmiedt, Mikael Ronström, Seif Haridi, and Jim Dowling. Hopsfs: Scaling hierarchical file system metadata using newsql databases, 2017.

[34] Andrew Jeffery, Heidi Howard, and Richard Mortier. Mutating etcd towards edge suitability, 2023.

[35] A. Katsarakis, V. Gavrielatos, M. Katebzadeh, A. Joshi, A. Dragojevic, B. Grot, and V. Nagarajan. Hermes: a fast, fault-tolerant and linearizable replication protocol, 2020.

[36] Peter Robinson. The local information cost of distributed graph spanners, 2024.

[37] Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. *arXiv preprint arXiv:1805.05404*, 2018.

[38] Jinkun Geng, Anirudh Sivaraman, Balaji Prabhakar, and Mendel Rosenblum. Nezha: Deployable and high-performance consensus using synchronized clocks, 2023.

[39] Mirco Richter. Crisis: Probabilistically self organizing total order in unstructured p2p networks, 2019.

[40] Wentao Cai, Haosen Wen, and Michael L. Scott. Transactional composition of nonblocking data structures, 2023.

[41] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma, 2021.

[42] Stefano De Angelis. Assessing security and performances of consensus algorithms for permissioned blockchains, 2018.

[43] Mayank Raikwar, Nikita Polyanskii, and Sebastian Müller. Sok: Dag-based consensus protocols, 2024.

[44] Pierre Sutra. On the correctness of egalitarian paxos, 2019.

[45] Chih-Wei Chien and Chi-Yeh Chen. Generalize synchronization mechanism: Specification, properties, limits, 2024.

[46] Kyeongmin Cho, Seungmin Jeon, and Jeehoon Kang. Practical detectability for persistent lock-free data structures, 2022.

[47] Andrew Lewis-Pye and Tim Roughgarden. Resource pools and the cap theorem, 2020.

[48] Alexey Gotsman and Sebastian Burckhardt. Consistency models with global operation sequencing and their composition (extended version), 2017.

[49] Wojciech Golab, Jeremy Hurwitz, Xiaozhou, and Li. On the k-atomicity-verification problem, 2013.

[50] Shir Cohen and Idit Keidar. Tame the wild with byzantine linearizability: Reliable broadcast, snapshots, and asset transfer, 2024.

[51] Paulo Sérgio Almeida and Carlos Baquero. Scalable eventually consistent counters over unreliable networks, 2013.

[52] Antonio Fernandez Anta, Chryssis Georgiou, Theophanis Hadjistasi, Nicolas Nicolaou, Efstathios Stavrakis, and Andria Trigeorgi. Fragmented objects: Boosting concurrency of shared large objects, 2021.

[53] Gal Amram, Lior Mizrahi, and Gera Weiss. Simple executions of snapshot implementations, 2015.

[54] Dylan Hutchison, Jeremy Kepner, Vijay Gadepally, and Bill Howe. From nosql accumulo to newsql graphulo: Design and utility of graph algorithms inside a bigtable database, 2016.

[55] Jeremy Kepner, Julian Chaidez, Vijay Gadepally, and Hayden Jansen. Associative arrays: Unified mathematics for spreadsheets, databases, matrices, and graphs, 2015.

[56] Tiago M. Vale, João Leitão, Nuno Preguiça, Rodrigo Rodrigues, Ricardo J. Dias, and João M. Lourenço. Lazy state determination: More concurrency for contending linearizable transactions, 2020.

[57] Ashish Tapdiya, Yuan Xue, and Daniel Fabbri. A comparative analysis of materialized views selection and concurrency control mechanisms in nosql databases, 2017.

[58] Mohammad Roohitavaf and Sandeep Kulkarni. Dkvf: A framework for rapid prototyping and evaluating distributed key-value stores, 2018.

[59] Christian Weilbach, Konrad Kühne, and Annette Bieniusa. Decoupling conflicts for configurable resolution in an open replication system, 2016.

[60] Yuqi Zhang, Yu Huang, Hengfeng Wei, and Xiaoxing Ma. Met: Model checking-driven explorative testing of crdt designs and implementations, 2022.

[61] Michael Elkin, Arnold Filtser, and Ofer Neiman. Distributed construction of light networks, 2019.

[62] Yeyuan Chen. The gap between greedy algorithm and minimum multiplicative spanner, 2024.

[63] David Eppstein and Hadi Khodabandeh. Optimal spanners for unit ball graphs in doubling metrics, 2022.

[64] Sariel Har-Peled, Manor Mendel, and Dániel Oláh. Reliable spanners for metric spaces. *ACM Transactions on Algorithms*, 19(1):1–27, 2023.

[65] Paolo Viotti and Marko Vukolić. Consistency in non-transactional distributed storage systems, 2016.

[66] Markus L. Schmid and Nicole Schweikardt. Refl-spanners: A purely regular approach to non-regular core spanners, 2024.

[67] Merav Parter, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. Local computation algorithms for spanners, 2019.

[68] T-H. Hubert Chan, Mingfei Li, and Li Ning. Incubators vs zombies: Fault-tolerant, short, thin and lanky spanners for doubling metrics, 2012.

[69] Arnold Filtser and Hung Le. Locality-sensitive orderings and applications to reliable spanners, 2022.

[70] Feodor F. Dragan and Muad Abu-Ata. Collective additive tree spanners of bounded tree-breadth graphs with generalizations and consequences, 2012.

[71] Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal, 2020.

[72] Deepthi Devaki Akkoorath, José Brandão, Annette Bieniusa, and Carlos Baquero. Global-local view: Scalable consistency for concurrent data types, 2017.

[73] Dan Alistarh, Trevor Brown, Justin Kopinsky, Jerry Z. Li, and Giorgi Nadiradze. Distributionally linearizable data structures, 2022.

[74] Alex Horn and Daniel Kroening. Faster linearizability checking via p-compositionality, 2015.

[75] Gal Sela, Maurice Herlihy, and Erez Petrank. Linearizability: A typo, 2021.

[76] Franz J. Hauck and Alexander Heß. Linearizability and state-machine replication: Is it a match?, 2024.

[77] Brijesh Dongol and John Derrick. Verifying linearizability: A comparative survey, 2015.

[78] Emanuele D'Osualdo, Azalea Raad, and Viktor Vafeiadis. The path to durable linearizability, 2022.

[79] Vassos Hadzilacos, Xing Hu, and Sam Toueg. On register linearizability and termination, 2021.

[80] Paulo Sérgio Almeida. A framework for consistency models in distributed systems, 2024.

[81] Armando Castañeda and Gilde Valeria Rodríguez. Asynchronous wait-free runtime verification and enforcement of linearizability, 2023.

[82] Andreas Haas, Thomas A. Henzinger, Andreas Holzer, Christoph M. Kirsch, Michael Lippautz, Hannes Payer, Ali Sezgin, Ana Sokolova, and Helmut Veith. Local linearizability, 2016.

[83] Uri Abraham. On the lazy set object, 2018.

[84] Artem Khyzha, Alexey Gotsman, and Matthew Parkinson. A generic logic for proving linearizability (extended version), 2016.

[85] Artem Khyzha, Mike Dodds, Alexey Gotsman, and Matthew Parkinson. Proving linearizability using partial orders (extended version), 2017.

[86] Roy Friedman, Michel Raynal, and François Taïani. Fisheye consistency: Keeping data in synch in a georeplicated world, 2015.

[87] Klaus v. Gleissenthall and Andrey Rybalchenko. An epistemic perspective on consistency of concurrent computations, 2013.

[88] Jeffrey Helt, Matthew Burke, Amit Levy, and Wyatt Lloyd. Regular sequential serializability and regular sequential consistency, 2021.

[89] Arik Rinberg and Idit Keidar. Intermediate value linearizability: A quantitative correctness criterion, 2020.

[90] Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch), 2017.

[91] Jad Hamza. On the complexity of linearizability, 2015.

[92] Ofer Grossman and Merav Parter. Improved deterministic distributed construction of spanners, 2017.

[93] Manuel Fernandez, David P. Woodruff, and Taisuke Yasuda. Graph spanners in the message-passing model, 2019.

Disclaimer:

SurveyX is an AI-powered system designed to automate the generation of surveys. While it aims to produce high-quality, coherent, and comprehensive surveys with accurate citations, the final output is derived from the AI's synthesis of pre-processed materials, which may contain limitations or inaccuracies. As such, the generated content should not be used for academic publication or formal submissions and must be independently reviewed and verified. The developers of SurveyX do not assume responsibility for any errors or consequences arising from the use of the generated surveys.