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 are fundamental to modern computing, providing frameworks for efficient management and processing of large-scale data. Their critical role in ensuring reliability and consistency across diverse applications is evident in their widespread adoption in contemporary software environments [1]. In cloud object storage systems, the need for integrity and consistency verification is heightened, particularly to protect against potentially malicious cloud providers [2].

The evolution of distributed databases has been propelled by challenges in achieving adaptivity and finality within blockchain protocols, underscoring their significance in decentralized settings [3]. These systems are also crucial for specifying concurrent behaviors, especially in large-scale data processing scenarios [4]. The persistence of data in concurrent structures poses significant challenges, as existing methods often depend on blocking mechanisms that hinder progress. This calls for the development of nonblocking approaches to enhance persistence [5].

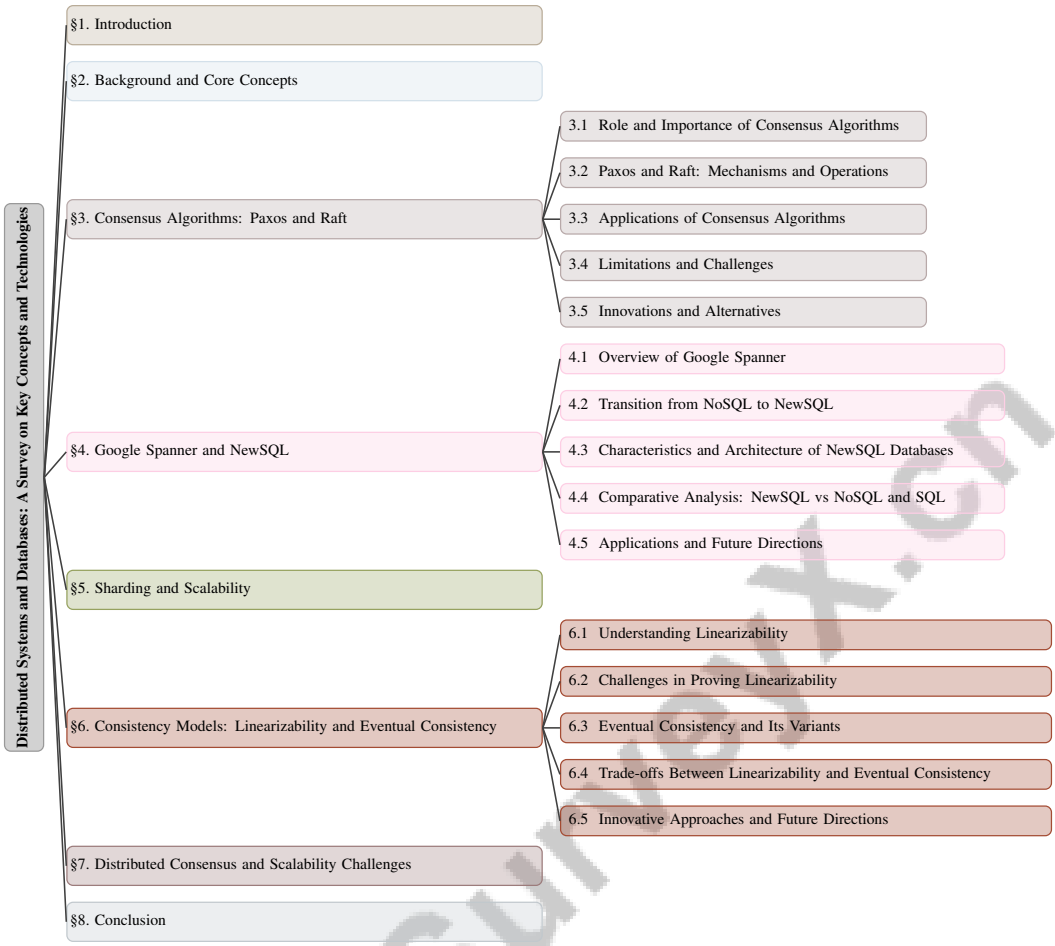


Figure 1: chapter structure

1.2 Key Concepts and Technologies Overview

Distributed systems and databases are supported by key concepts and technologies essential for addressing modern computing challenges. The CAP theorem provides a foundational framework for understanding the trade-offs between consistency, availability, and partition tolerance, which are critical in the design and operation of distributed databases [6]. The ACID and BASE models further illustrate the spectrum of approaches to transaction management and reliability within database systems [7].

Evolutionary data systems (EDS) represent a significant advancement, adapting their architecture to evolving workloads and application requirements, thus overcoming the limitations of static systems [8]. Polyglot data management utilizes various data stores to provide versatile solutions for diverse use cases [9], while polystore databases integrate SQL, NoSQL, and NewSQL databases to optimize data exchange and query execution across heterogeneous systems [10].

Consensus algorithms such as Paxos and Raft are pivotal in ensuring agreement on data values across distributed systems, maintaining consistency among nodes. These mechanisms extend to blockchain technologies, which facilitate decentralized architectures and consensus models to reduce operational costs in applications like mobile peer-to-peer games [11]. Strong linearizability is also examined for its implications in concurrent programs, particularly concerning hyper-properties preserved through primitives with consensus number 2 [12].

The complexities introduced by network partitions and failures necessitate new abstractions and models [13]. Concepts such as fork-linearizability and the Conflict-free Operation verification Protocol (COP) are essential for verifying consistency in remote, untrusted environments [14]. Additionally, the Matrix Event Graph (MEG) is identified as a replicated data type that enhances resilience and

scalability in decentralized applications [15]. These foundational concepts and technologies will be further explored in subsequent sections to provide a comprehensive understanding of their roles and implications in distributed systems and databases.

1.3 Structure of the Survey

This survey is systematically organized to thoroughly explore distributed systems and databases, emphasizing the key concepts and technologies that define this domain. It begins with an overview of distributed systems and databases, detailing their essential functions in contemporary computing environments. The interplay between database technology and emerging innovations like blockchain is examined, illustrating how traditional database features such as ACID compliance and real-time analytics enhance blockchain capabilities. The implications of the CAP theorem on distributed systems are discussed, including its limitations and alternative frameworks for understanding trade-offs between consistency and availability. This foundational discussion sets the stage for an in-depth exploration of storage systems utilized in high-performance computing (HPC) and big data analytics (BDA), analyzing their architectures, data consistency mechanisms, and fault tolerance strategies, ultimately leading to a deeper understanding of the core concepts and technologies underpinning modern distributed systems [16, 17, 18, 7, 19].

Subsequent sections delve into background and core concepts, providing detailed explanations of the CAP theorem, ACID and BASE models, and their implications for database transactions, alongside the significance of consistency, availability, and partition tolerance.

The survey continues with a focus on consensus algorithms, particularly Paxos and Raft, discussing their mechanisms, applications, and limitations. This is followed by an examination of Google Spanner and NewSQL databases, highlighting their characteristics and the transition from NoSQL to NewSQL. The discussion addresses sharding and scalability, detailing how these concepts enhance performance and reliability in distributed systems.

A comparison of consistency models, such as linearizability and eventual consistency, analyzes their trade-offs and use cases. Finally, the survey examines challenges in distributed consensus and scalability, offering insights into innovative algorithms and strategies for overcoming 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 the realm of distributed databases, outlining the inherent trade-offs among Consistency, Availability, and Partition Tolerance [17]. It posits that a distributed system cannot simultaneously guarantee all three properties, necessitating prioritization based on system requirements [3]. Consistency, ensuring uniform data state across nodes, is essential for data integrity in concurrent settings [20], yet it often conflicts with availability and partition tolerance, which are crucial for service continuity during network partitions [9, 21]. Consequently, systems may adopt the BASE model for eventual consistency when high availability is prioritized [2].

The theorem's implications extend to consensus algorithms and replication protocols, particularly in maintaining linearizability amid Byzantine failures [20]. Its principles guide the development of replication protocols that balance throughput and latency, especially in inter-data center communications [21]. Extensions like the CALM theorem propose that monotonic programs can achieve all three properties, enriching discussions on consistency, availability, and latency trade-offs [1]. Thus, the CAP theorem remains integral to architectural decisions and influences both theoretical frameworks and practical implementations in distributed computing [22].

2.2 ACID vs BASE Models

ACID (Atomicity, Consistency, Isolation, Durability) and BASE (Basically Available, Soft state, Eventually consistent) represent contrasting paradigms in database transaction management, each

impacting distributed systems' reliability and performance. ACID transactions ensure complete execution or rollback of operations, maintaining data integrity even during failures, which is critical for applications requiring strict consistency [23]. In contrast, the BASE model prioritizes availability and partition tolerance, allowing temporary inconsistencies until eventual consistency is reached [24], which is advantageous in systems where high availability is crucial despite challenges in maintaining algorithmic correctness [9].

The dichotomy between ACID and BASE reflects broader trade-offs between consistency and availability. ACID offers strong guarantees necessary for data integrity but incurs performance overheads in distributed environments [17]. BASE, focusing on availability, supports scalable and resilient systems at the risk of temporary inconsistencies [9]. Innovations like checkpointing and rollback algorithms in distributed shared memory systems aim to reconcile these trade-offs by balancing correctness and performance [25]. The CALM theorem also provides insights into achieving consistency, availability, and partition tolerance simultaneously [1]. Ultimately, the choice between ACID and BASE depends on specific application requirements, with ACID favored for immediate consistency and BASE for scalability and availability [17].

2.3 Consistency and Fault Tolerance

Consistency and fault tolerance are essential in distributed systems, ensuring reliable operations amid failures. Consistency guarantees uniform data state across nodes, crucial for data integrity in concurrent operations [26]. Advanced models like distributional and fragmented linearizability enhance relaxed concurrent data structures' effectiveness under high concurrency [20]. Fault tolerance enables correct operation despite failures, including Byzantine faults [2], necessitating robust consistency models.

The interplay between consistency and fault tolerance is complex, as high consistency demands extensive synchronization, potentially hindering performance in high-latency environments [19]. Innovative strategies like the SWARM model reduce operational roundtrips, minimizing latency while preserving strong consistency and liveness. Conflict resolution mechanisms, such as CDVCS, enhance flexibility and robustness in maintaining consistency across replicas [5]. Data replication is crucial for reliability and fault tolerance, supporting consistent delivery orders through properties like MS-Ordering, which govern message interactions [1].

By addressing synchronization, concurrency, and adversarial conditions, distributed systems can balance consistency and availability during network partitions. Strategies like Just-Right Consistency and Conflict-Free Replicated Data Types (CRDTs) facilitate asynchronous updates while ensuring correctness. The CALM theorem further elucidates when coordination is necessary, enabling coordination-free applications to maintain consistent semantics, enhancing fault tolerance and operational reliability in dynamic environments [1, 27, 28, 29].

2.4 Partition Tolerance and Scalability

Partition tolerance is fundamental in distributed systems, ensuring correct functionality during network partitions, crucial for large-scale systems where disruptions are common [17]. This capability supports scalability, allowing systems to handle increased loads and expand across networks without performance degradation.

Scalability is often achieved through innovative data structures and algorithms optimizing communication and computation during network partitions. The nbMontage system exemplifies this by transforming nonblocking concurrent data structures into persistent structures that maintain high performance and low latency [5]. The congested clique model illustrates how partition tolerance can enhance communication and computation efficiency [17], minimizing communication rounds and improving scalability.

Challenges in scalability arise from differing mathematical foundations and operational semantics across database systems, creating bottlenecks [30]. Centralized metadata management in distributed file systems can hinder performance as data volume grows, necessitating scalable solutions that distribute metadata management [31]. Categorizing methods based on sensitivity to network delay—differentiating delay-sensitive (O(d)) from delay-independent (O(1)) operations—provides insights into optimizing scalability under varying network conditions [17].

Partition tolerance is essential for achieving scalability in distributed systems, directly influencing trade-offs between consistency and availability, especially during network partitions, where maintaining performance and responsiveness is paramount for large-scale applications [32, 33, 27, 17, 19]. By ensuring effective operation despite disruptions and leveraging innovative algorithms and data structures, distributed systems can achieve robust performance and scalability, meeting modern computing demands.

3 Consensus Algorithms: Paxos and Raft

Category	Feature	Method
Role and Importance of Consensus Algorithms	Data Consistency Assurance	VICOS[2], ABV[34], nbM[5], CLC[3]
Paxos and Raft: Mechanisms and Operations	Replication and Consistency	N/A[35], HMS[36]
Applications of Consensus Algorithms	Consistency and Performance Efficient Operations Fairness and Transparency	WFSLI[12] CCSC[37] PoT[11]
Limitations and Challenges	Communication and Efficiency Execution and Order Management Consistency and Atomicity	LIC[38], CCSC[39] N/A[40], Crisis[41] NBTC[42]
Innovations and Alternatives	Adaptive Protocol Strategies	EFP[43]

Table 1: This table provides a comprehensive summary of various categories, features, and methods associated with consensus algorithms, particularly focusing on their roles in ensuring data consistency, replication, and performance in distributed systems. It also highlights the applications, limitations, and innovative alternatives to traditional consensus mechanisms like Paxos and Raft, offering insights into their operational dynamics and challenges.

In distributed systems, consensus among nodes is critical for maintaining reliability and consistency, leading to the development of algorithms like Paxos and Raft, which are renowned for their distinct mechanisms and operational features. Understanding these algorithms is crucial for appreciating their roles in preserving system integrity and performance. As illustrated in **??**, the hierarchical structure of consensus algorithms highlights their fundamental roles in ensuring data consistency and reliability. This figure emphasizes the mechanisms and operations of Paxos and Raft, as well as their applications in blockchain and distributed databases. Table 1 presents a detailed summary of the key aspects of consensus algorithms, emphasizing their importance in distributed systems, and providing insights into their methods and applications. Table 4 provides a comparative overview of the Paxos and Raft consensus algorithms, focusing on their distinct operational features and complexity levels. This section explores the significance of consensus algorithms in maintaining data integrity and consistency, followed by an examination of Paxos and Raft, highlighting their unique characteristics and performance implications in contexts such as permissioned blockchains and high-performance cloud applications [44, 45, 41, 40].

3.1 Role and Importance of Consensus Algorithms

Method Name	**System Consistency**	**Performance Balance**	**Application Domains**
VICOS[2]	Fork-linearizability	Low Overhead	Cloud Storage
ABV[34]	Consistency Policies	Consistency And Performance	Cloud Storage, Blockchain
nbM[5]	Consensus Algorithms	Latency Consistency Balance	Cloud Storage Blockchain
CLC[3]	Byzantine Fault Tolerant	Latency And Consistency	Blockchain Technologies

Table 2: Overview of consensus algorithms and their attributes, highlighting their system consistency, performance balance, and application domains in cloud storage and blockchain technologies. The table includes methods such as VICOS, ABV, nbM, and CLC, each contributing to the understanding and advancement of distributed systems through varied consistency policies and performance metrics.

Consensus algorithms are fundamental in distributed systems, ensuring agreement on data values across nodes, which is crucial for consistency and reliability, especially in environments prone to failures or malicious actions [19]. They are vital for executing distributed transactions correctly, supporting system recovery from failures, and managing concurrent operations effectively. The balance between latency and consistency is essential, as highlighted by the critical role of understanding latency in distributed systems [19]. This balance is crucial for maintaining system integrity and efficiency under unpredictable network conditions.

As illustrated in Figure 2, the pivotal role of consensus algorithms in distributed systems is emphasized, highlighting their importance for consistency, reliability, and concurrent operations. The figure also showcases applications in cloud storage and blockchain technologies, alongside recent research innovations such as the CAL theorem and the VICOS protocol, which enable multiple clients to verify data integrity and consistency without fully trusting the server [2]. The development of semantics and specifications for concurrent libraries in weakly-consistent, replicated settings underscores the necessity of consensus algorithms for correct and efficient operation [34]. Frameworks categorizing research on polyglot data management emphasize the diverse roles of consensus algorithms in distributed systems [9].

Table 2 presents a comprehensive analysis of various consensus algorithms, detailing their system consistency, performance balance, and application domains, thereby elucidating their critical role in distributed systems. Innovations like nbMontage illustrate the benefits of nonblocking data structures over traditional locking mechanisms, enhancing performance and resilience to deadlocks [5]. In blockchain technologies, consensus algorithms play a critical role, as seen in the checkpointed longest chain protocol, emphasizing their importance in achieving agreement among distributed systems [3].

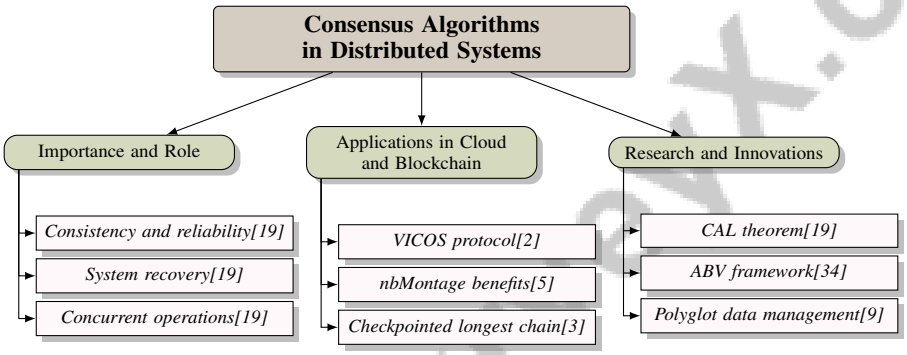


Figure 2: This figure illustrates the pivotal role of consensus algorithms in distributed systems, highlighting their importance for consistency, reliability, and concurrent operations. It also showcases applications in cloud storage and blockchain technologies, alongside recent research innovations such as the CAL theorem, VICOS protocol, and polyglot data management.

3.2 Paxos and Raft: Mechanisms and Operations

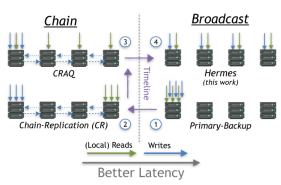
Paxos and Raft are key consensus algorithms ensuring nodes agree on data values, maintaining consistency and reliability by addressing challenges like network failures and partitions. They navigate trade-offs between consistency and availability based on network disruptions, as demonstrated by frameworks like Crisis, which adapts to varying conditions in unstructured peer-to-peer networks. Alternative consensus methods, such as Proof-of-Authority and Practical Byzantine Fault Tolerance, are evaluated for their security and performance in permissioned blockchains, highlighting the importance of robust analysis under real-world conditions [44, 17, 19, 41, 46].

Paxos, introduced by Leslie Lamport, employs a formal approach to consensus through proposing, voting, and committing phases among nodes. It elects a proposer to suggest a value to a quorum of acceptors; if a majority agree, the value is chosen and communicated to learners, ensuring convergence to the same state. Despite its high fault tolerance, Paxos is often criticized for complexity and performance overhead [45]. Egalitarian Paxos (EPaxos) addresses these challenges by using two ballot variables to enhance safety and consistency [45].

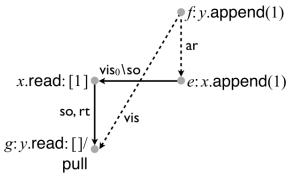
Raft simplifies consensus with a strong leader model, where a leader manages log replication and ensures consistency across followers. The leader handles client requests, appending entries to its log and replicating them to followers. Once a majority of followers acknowledge the entries, they are committed, ensuring all nodes reflect the same state. Raft's design prioritizes understandability and ease of implementation, making it a favorable choice for practical deployments [47]. Raft also integrates detectable operations into persistent lock-free data structures for efficient recovery and consistency after crashes [48].

Paxos and Raft differ significantly in their approaches. Paxos uses a decentralized model with multiple proposers and acceptors, whereas Raft employs centralized leadership. These differences impact their performance and suitability for various applications; Paxos is preferred in high-fault-tolerance environments, while Raft is favored for its simplicity and accessibility [45, 16, 29, 40].

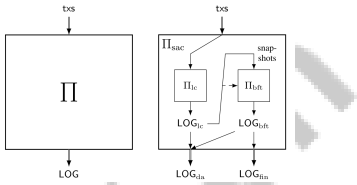
Evaluating these algorithms under various conditions reveals their strengths and limitations. For example, Hermes achieves high throughput and low latency while providing linearizable reads and writes, illustrating potential optimizations in consensus mechanisms [36]. Adaptations of etcd utilizing Conflict-free Replicated Data Types (CRDTs) improve performance in edge environments [35].



(a) Chain-Based Data Replication with Hermes[36]



(b) A Graphical Representation of a Program Flow[49]



(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."[43]

Figure 3: Examples of Paxos and Raft: Mechanisms and Operations

The illustrations in Figure 3 provide practical examples of the operational dynamics of consensus algorithms, particularly Paxos and Raft. The first example, "Chain-Based Data Replication with Hermes," contrasts traditional replication methods with Hermes, highlighting enhanced data replication efficiency. The second example, "A Graphical Representation of a Program Flow," illustrates the control flow and variable interactions critical for understanding consensus operations. Finally, the block diagram featuring a central square labeled "" and its connections to components "sac" and "bft" showcases the structural interactions within a consensus system. Together, these visual aids underscore the complexity and sophistication of Paxos and Raft algorithms in ensuring reliable data replication in distributed systems.

3.3 Applications of Consensus Algorithms

Method Name	Application Domains	Consistency Mechanisms	Performance Trade-offs
PoT[11]	Turn-based Games	Proof-of-Turn	Scalability And Trust
WFSLI[12]	Concurrent Programming	Strong Linearizability	Wait-free Implementations
CCSC[37]	Distributed Networks	Localized Algorithms	Round Complexity
CCSC[39]	Distributed Computation	Hitting-set Algorithm	Round Complexity

Table 3: Overview of consensus algorithms with their application domains, consistency mechanisms, and performance trade-offs. The table highlights various methods, illustrating their applicability in diverse fields such as turn-based games and distributed networks, while detailing the specific consistency strategies and associated performance considerations.

Consensus algorithms are integral to real-world distributed systems, providing the foundation for agreement among nodes and ensuring data consistency across diverse applications. They are employed in various fields, including distributed systems and concurrent programming, addressing challenges such as the trade-offs between consistency and availability during network partitioning, as highlighted by the CAL theorem, and the nuanced requirements of task and object specifications in distributed computing [4, 19]. Table 3 provides a comprehensive overview of consensus algorithms, detailing their application domains, consistency mechanisms, and the inherent performance trade-offs involved.

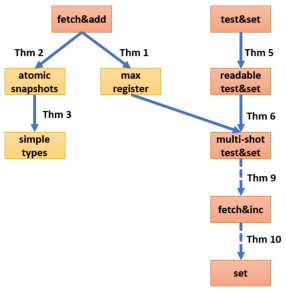
A significant application of consensus algorithms is in blockchain technologies, where they maintain a decentralized and tamper-proof ledger. The Proof of Turn (PoT) consensus mechanism exemplifies this application by reducing server costs while ensuring a fair and transparent gaming environment,

vital for player retention [11]. PoT's ability to provide equitable access within a blockchain network demonstrates the versatility of consensus algorithms in fostering trust and efficiency.

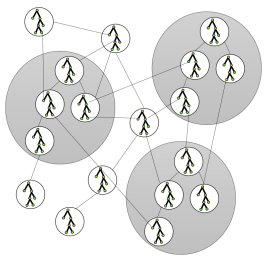
In distributed databases, consensus algorithms coordinate transactions and maintain data integrity across multiple nodes, crucial for systems requiring high availability and fault tolerance. They facilitate reliable data replication and synchronization by balancing the trade-offs outlined in the CAP theorem, which posits that a system can achieve either consistency or availability during network partitioning, but not both. Advanced mechanisms, such as centralized and decentralized coordination, enable tailored solutions that maintain bounded consistency or availability based on specific application needs. Methodologies like Conflict-Free Replicated Data Types (CRDTs) and Transactional Causal Consistency allow asynchronous updates without sacrificing correctness, enhancing scalability while ensuring data integrity. Systems like Antidote exemplify these principles, supporting industrial-grade applications validated in extensive experimental settings across multiple geo-distributed data centers [29, 19].

Furthermore, consensus algorithms are applied in verifying distributed transactions, facilitating the analysis of transaction histories to ensure atomicity and correctness. Techniques such as Forward and Backward Zone (FZF) analysis break down transaction histories, allowing efficient verification of atomicity properties in distributed systems [50]. This role enhances the reliability and correctness of distributed transactions, critical for applications requiring stringent consistency guarantees.

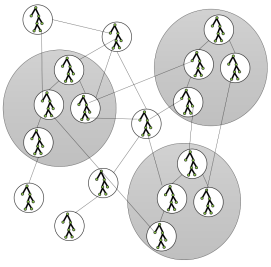
In cloud storage systems, consensus algorithms maintain data consistency and integrity across geographically distributed data centers. They ensure reliable access to data despite potential network partitions or latency issues, adhering to varying consistency and availability requirements. Different consensus protocols, such as Proof-of-Authority and Practical Byzantine Fault Tolerance, address specific challenges in permissioned blockchain environments, offering varying levels of security and performance essential for data integrity in real-world applications. Advanced systems like PCAP dynamically adapt to changing network conditions to optimize both latency and consistency, underscoring the necessity of tailored consensus mechanisms in managing distributed storage solutions [32, 29, 33, 44, 19]. By coordinating updates and resolving conflicts among replicas, these algorithms enable cloud services to deliver consistent and reliable data access, highlighting the significance of consensus in maintaining the performance and reliability of cloud-based services.



(a) The image represents a flowchart illustrating the relationships between different theorems and concepts in a mathematical or logical framework.[12]



(b) The image depicts a network of interconnected nodes[37]



(c) Network of interconnected trees[39]

Figure 4: Examples of Applications of Consensus Algorithms

The illustrations in Figure 4 highlight the crucial role of consensus algorithms in distributed systems, ensuring that multiple nodes can agree on a single data value despite failures. The first image presents a flowchart that emphasizes the relationships between various theorems and concepts within a logical framework, essential for consensus. The second image captures a network of interconnected nodes, symbolizing the complexity of distributed systems where algorithms like Paxos and Raft operate to maintain consistency. The third image depicts a network of interconnected trees, illustrating the hierarchical structures that consensus algorithms navigate to ensure synchronized operations. These visual examples underscore the significance of consensus algorithms in maintaining order and consistency within dynamic environments [12, 37, 39].

AI-generated, for reference only.

3.4 Limitations and Challenges

Implementing consensus algorithms such as Paxos and Raft in distributed systems presents several limitations and challenges, particularly regarding performance, consistency, and fault tolerance. A major challenge is achieving optimal resilience against Byzantine failures, which can severely undermine the reliability of decentralized systems. Proposed methods to enhance resilience often involve complex mechanisms to ensure consistency and security, emphasizing the need for advanced strategies to maintain system reliability under adversarial conditions [51].

While Paxos and Raft are designed to handle node failures and network partitions, they may struggle under extreme adversarial conditions or high levels of network partitioning, where maintaining consistency and availability becomes increasingly difficult. Their performance is often constrained by leader-based architectures, which can become bottlenecks during high-load scenarios. Nezha addresses this by separating execution from commitment, thus reducing the leader's burden and improving performance [40]. This separation is crucial for enhancing throughput and reducing latency, yet it requires careful management to ensure synchronization and consistency across nodes.

Another challenge is constructing spanner algorithms, where achieving both time- and communication-optimality is fundamentally difficult. This limitation affects the efficiency of consensus algorithms, which rely on effective communication and coordination among nodes to achieve agreement. The difficulty in constructing efficient spanners is compounded by the need to balance communication overhead with the robustness of consensus mechanisms [38]. Existing methods in spanner construction further highlight the challenges of achieving efficient consensus and scalability, particularly in high-communication-overhead environments [39].

The limitations of Proof of Authority (PoA) algorithms, which offer better performance in latency, underscore the trade-offs between performance and consistency guarantees. While these algorithms excel in low-latency environments, they often lack the robust consistency provided by algorithms like Practical Byzantine Fault Tolerance (PBFT), particularly under adversarial conditions [44]. This trade-off reflects broader challenges in achieving both efficiency and reliability in distributed consensus protocols.

Achieving total order in dynamic environments poses significant challenges, requiring a balance between communication overhead and fault tolerance. Crisis provides a robust mechanism for achieving total order with lower communication overhead, yet it still faces challenges in maintaining fault tolerance in highly dynamic settings [41]. This underscores the need for innovation in consensus algorithm design to accommodate the dynamic nature of distributed systems.

Lastly, identifying linearization points in complex data structures and ensuring atomic execution of critical memory accesses complicates the implementation of consensus algorithms. These challenges necessitate advanced techniques for effectively managing consistency and performance in distributed systems [42]. The fundamental trade-off in permissionless blockchain protocols, where systems can either be adaptive or possess finality but not both, further complicates the landscape of distributed consensus [52].

3.5 Innovations and Alternatives

Recent advancements in consensus algorithms have introduced innovative approaches and alternatives to traditional models like Paxos and Raft, aiming to enhance performance, resilience, and adaptability in distributed systems. One notable innovation is the development of ebb-and-flow protocols, which offer a dynamic approach to consensus by adjusting operational parameters based on network conditions and adversarial strategies. These protocols are designed to provide both availability and finality, adapting to changing environments to maintain robust performance [43].

Future research could focus on enhancing the resilience of ebb-and-flow protocols against sophisticated adversarial strategies. Integrating these protocols into existing blockchain frameworks could lead to greater adaptability and security, addressing traditional consensus algorithms' limitations in dynamic and adversarial conditions [43]. This integration could pave the way for more robust blockchain systems capable of maintaining consensus and performance even under challenging network scenarios.

Additionally, alternative approaches to consensus are exploring hybrid models that combine elements of both Paxos and Raft, leveraging the strengths of each to optimize performance and reliability.

These hybrid models aim to balance the decentralized nature of Paxos with Raft's leader-based efficiency, providing a more flexible and scalable solution for distributed systems. Features such as dynamic leader election and adaptive quorum sizes can significantly improve fault tolerance and minimize latency. This advancement addresses the CAP theorem's inherent trade-offs—where consistency, availability, and partition tolerance must be balanced—while offering a robust framework for evaluating the performance and security of consensus algorithms in real-world applications, particularly in permissioned blockchain environments. Such enhancements lay the groundwork for future research and development aimed at optimizing consensus mechanisms for decentralized systems [44, 41, 47, 19].

Feature	Paxos	Raft
Mechanism Type	Decentralized	Centralized
Leadership Model	Multiple Proposers	Strong Leader
Complexity Level	High Complexity	Low Complexity

Table 4: Comparison of Paxos and Raft Consensus Algorithms: An analysis of the key features distinguishing Paxos and Raft, highlighting their mechanism types, leadership models, and complexity levels. This comparison elucidates the fundamental operational differences and implications for their application in distributed systems.

4 Google Spanner and NewSQL

To comprehend advancements in distributed database technology, Google Spanner exemplifies the NewSQL paradigm. This section provides an overview of Spanner's architecture and innovative features, setting the stage for a deeper exploration of its capabilities and implications in modern database management.

4.1 Overview of Google Spanner

Google Spanner represents a significant shift in distributed database design, offering a globally distributed system that combines the strong consistency of traditional relational databases with the horizontal scalability of NoSQL systems. Its architecture leverages the TrueTime API, providing a globally synchronized clock that enables external consistency, ensuring transactions are executed in real-time order across geographically distributed nodes [21]. This design addresses scalability and consistency challenges through a tiered system similar to Handoff Counters, facilitating efficient state management without a centralized sequencer, and a fragmented object approach akin to COBFS, enhancing concurrency and scalability by treating large datasets as manageable blocks [53, 54].

Spanner's adaptive learning mechanisms evolve based on operational feedback, ensuring performance and reliability in large-scale dynamic environments [55]. Its deployment across multiple data centers, as demonstrated by the LEGOStore prototype on Google Cloud Platform, showcases its optimization of cost savings and latency while maintaining a linearizable data store [21]. Furthermore, as a polystore system, Spanner integrates multiple data management paradigms to provide versatile data integration strategies [9]. Protocols like VICOS enhance the reliability of cloud object storage systems by maintaining a consistent view of operations across multiple clients [2].

4.2 Transition from NoSQL to NewSQL

The transition from NoSQL to NewSQL databases signifies an evolution in database management systems, addressing the limitations of traditional RDBMS and NoSQL systems in handling online transaction processing (OLTP) workloads within big data contexts [18]. While NoSQL databases emerged to tackle scalability challenges, they often compromise ACID properties critical for transactional consistency [7]. NewSQL databases bridge the gap by efficiently managing high-velocity data while supporting complex queries [18], as illustrated by the integration of algorithms into systems like Apache Accumulo through Graphulo [56].

A key aspect of this transition is the use of associative arrays as a mathematical framework, optimizing data processing [10]. The D4M software system exemplifies this application, enhancing data integration and processing efficiency [57]. Additionally, the emergence of evolutionary data systems allows NewSQL databases to dynamically adapt to incoming data and queries [8]. Distributed metadata services like HopsFS further illustrate NewSQL's advantages in efficient metadata management and scalability [31].

Figure 5 illustrates the transition from NoSQL to NewSQL databases, highlighting the challenges associated with NoSQL systems, the advantages offered by NewSQL, and the innovative approaches facilitating this transition. This figure serves to visually reinforce the discussion of how NewSQL addresses the shortcomings of its predecessor, thereby enhancing our understanding of the current landscape in database management.

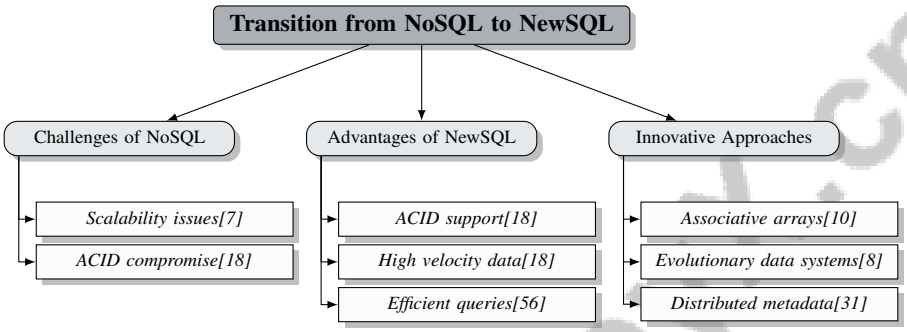
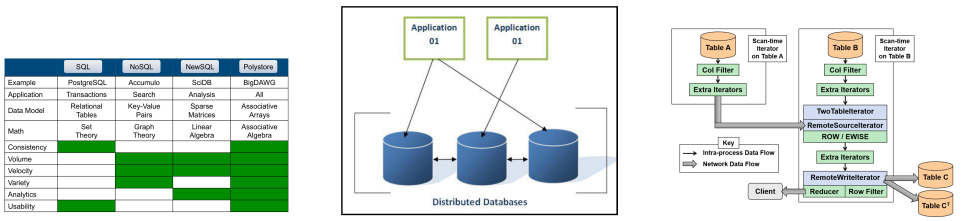


Figure 5: This figure illustrates the transition from NoSQL to NewSQL databases, highlighting the challenges associated with NoSQL systems, the advantages offered by NewSQL, and the innovative approaches facilitating this transition.

4.3 Characteristics and Architecture of NewSQL Databases

NewSQL databases are designed to deliver NoSQL scalability while maintaining the ACID guarantees of traditional SQL databases. They support high-velocity data processing and complex transactional workloads, making them suitable for data-intensive applications. A defining feature is their reliance on associative array algebra, providing a unified framework for operations across SQL, NoSQL, and NewSQL databases, optimizing data processing and reducing interaction complexity [30].

The architecture of NewSQL databases incorporates advanced concurrency control mechanisms, such as Lazy State Determination (LSD), enhancing performance by minimizing overhead in maintaining strong consistency across distributed nodes [58]. This approach allows NewSQL databases to achieve high throughput and low latency in distributed environments. NewSQL systems also offer advantages over traditional NoSQL and SQL databases, such as improved join performance and query expressiveness, particularly evident in materialized views, enabling real-time analytics and data-driven decision-making [59]. The integration of qualitative and quantitative analyses into NewSQL methodologies enhances operational efficiency, as demonstrated in blockchain applications, with the Proof of Turn (PoT) mechanism illustrating NewSQL's adaptability in supporting diverse use cases while reducing operational costs [11].



(a) NoSQL Database Systems Comparison[10]

(b) Distributed Databases[18]

(c) Scalable MapReduce Framework[56]

Figure 6: Examples of Characteristics and Architecture of NewSQL Databases

As shown in Figure 6, the emergence of NewSQL databases, characterized by innovative architectures and capabilities, marks a significant evolution in database management systems, bridging the gap between traditional SQL databases and the scalability of NoSQL systems. Google Spanner exemplifies these characteristics, designed to deliver ACID guarantees while providing the horizontal scalability of NoSQL systems. The accompanying figures illustrate various aspects of NewSQL databases, highlighting their ability to handle large volumes of data with high velocity and variety. By integrating the strengths of both SQL and NoSQL paradigms, NewSQL databases are positioned to meet the demands of modern data-intensive applications, offering robust solutions for scalability and consistency challenges in distributed computing environments [10, 18, 56].

4.4 Comparative Analysis: NewSQL vs NoSQL and SQL

Benchmark	Size	Domain	Task Format	Metric
BPC[44]	1,000,000	Blockchain	Consensus Evaluation	Throughput, Latency
CRB[60]	10,000	Logical Reasoning	Multi-step Reasoning	Accuracy, F1-score

Table 5: Table 1 provides a comprehensive summary of representative benchmarks used to evaluate various database systems. It includes information on the size, domain, task format, and metrics employed for performance assessment, offering insights into the diverse testing environments and criteria utilized in database research.

The comparative analysis of NewSQL, NoSQL, and traditional SQL databases highlights the distinct strengths and weaknesses inherent in each system. Table 5 presents a detailed overview of benchmarks utilized in the comparative analysis of NewSQL, NoSQL, and SQL databases, highlighting their respective domains and evaluation metrics. NoSQL systems emerged to address traditional SQL databases' limitations, particularly regarding scalability and schema flexibility. These systems prioritize performance and horizontal scaling, making them suitable for applications requiring rapid data ingestion across distributed architectures. However, this often comes at the expense of the strong consistency and security guarantees provided by traditional RDBMS [18].

Traditional SQL databases ensure strong transactional integrity through ACID properties, essential for precise data management in environments where data integrity is critical. In contrast, while NewSQL and NoSQL technologies are designed for scalability and large data volumes, they may sacrifice some ACID compliance for performance and flexibility, making traditional SQL databases preferable for scenarios demanding stringent data reliability [29, 59, 22, 18, 7]. These systems excel in applications where data accuracy and consistency are paramount, such as financial transactions and enterprise resource planning, although SQL databases' scalability limitations can hinder modern applications demanding high throughput and low latency.

NewSQL databases seek to reconcile the scalability of NoSQL systems with the consistency guarantees of SQL databases. They leverage 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 the scalability of NoSQL and the reliability of traditional RDBMS. By integrating associative array algebra and advanced data management techniques, NewSQL systems optimize data processing and efficiently support complex transactional workloads [18].

4.5 Applications and Future Directions

NewSQL databases are increasingly vital in distributed systems, offering robust solutions for applications requiring high scalability and strong consistency. They are particularly suited for OLTP, real-time analytics, and cloud services, efficiently managing complex transactional workloads. By leveraging associative array algebra, NewSQL databases optimize data processing and support real-time decision-making, making them indispensable in data-intensive environments [30].

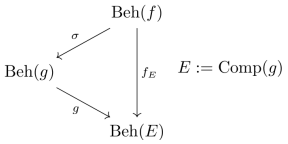
In cloud computing, NewSQL databases provide scalable and reliable infrastructure for distributed applications, ensuring data consistency and availability across geographically dispersed data centers. This capability is essential for maintaining seamless user experiences and supporting global operations, as evidenced by their integration in cloud-based services that require robust data management solutions to handle increasing data volumes and velocities [61]. NewSQL systems' adaptability to diverse workloads and their performance under challenging network conditions further underscore

their applicability in edge computing environments, where local-first operations are crucial for reducing latency and enhancing user experiences [35].

Future research in NewSQL databases is poised to explore several promising directions. Expanding benchmarks to include additional reasoning tasks and integrating multimodal data may enhance evaluation methods for NewSQL systems [60]. Additionally, employing model-checking driven explorative approaches across a broader range of distributed systems and heuristic strategies could optimize the model-checking process [62].

Advancements in network design and distributed systems, such as fault-tolerant spanners, offer insights into improving the robustness and scalability of NewSQL architectures [63]. Exploring approximately universal optimality in weighted graphs and other classes of spanners could enhance network design and graph algorithms, indicating potential applications in optimizing NewSQL databases [64]. Furthermore, the exploration of hybrid systems supporting both locking and nonblocking structures could lead to comprehensive solutions for transactional memory, addressing current limitations and expanding the applicability of NewSQL databases [5].

(a) Graph Operations in SQL, NoSQL, and NewSQL[10]



(b) A diagram illustrating the composition of functions and their behavior[65]

GraphBLAS Kernel	Graphulo Implementation
BuildMatrix ($\oplus$)	Accumulo BatchWriter
ExtracTuples	Accumulo BatchScanner
MxM ($\oplus$, $\otimes$)	TwoTableIterator ROW mode, performing $\mathbf{A}^{\mathsf{T}}\mathbf{B}$
EwiseMult ($\otimes$)	TwoTableIterator EWISE mode
EwiseAdd ($\oplus$)	Similar to EwiseMult, with non-matching entries
Extract	Row and column filtering
Apply (f)	Extra Iterators
Assign	Apply with a key-transforming function
Reduce ($\oplus$)	Reducer module on RemoteWriteIterator
Transpose	Transpose option on RemoteWriteIterator

(c) GraphBLAS Kernel Implementation[56]

Figure 7: Examples of Applications and Future Directions

As shown in Figure 7, the exploration of Google Spanner and NewSQL in database systems highlights the evolving landscape of data management technologies, focusing on their applications and future directions. Illustrated through figures, this example delves into the comparative capabilities of SQL, NoSQL, and NewSQL, particularly in handling graph operations. The first figure juxtaposes these systems, showcasing how SQL employs set operations, NoSQL utilizes graph operations, and NewSQL leverages matrix operations for effective data management. Additionally, a diagram elucidates the composition of functions, emphasizing the transformation and flow of data through various behavioral functions, signifying the intricate nature of modern data processing techniques. The GraphBLAS Kernel Implementation demonstrates practical applications of these kernels in Graphulo, providing insights into operational efficiencies and potential advancements in data processing. Collectively, these visual and descriptive elements underscore the innovative approaches and future possibilities within the realm of database technologies, as exemplified by Google Spanner and NewSQL [10, 65, 56].

5 Sharding and Scalability

Scalability is pivotal in distributed systems, influencing data management efficiency and performance. Sharding, which divides data into smaller, manageable segments, stands out as a key strategy for achieving scalability. This technique not only distributes workload across multiple nodes but also enhances fault tolerance and resource utilization. The following subsection delves into the core principles of sharding in distributed systems.

5.1 Concept of Sharding in Distributed Systems

Sharding partitions databases into smaller segments, allowing distributed systems to efficiently manage increased loads by distributing data across nodes, optimizing resource utilization, and enhancing performance. This approach is akin to constructing lightweight bounded-degree spanners, facilitating efficient communication and computation in distributed environments [66]. By dividing a database into independently manageable shards, systems can handle large data volumes and high transaction rates without performance loss. Distributing shards across servers improves load balancing and fault tolerance, ensuring that failures in one shard do not affect the entire system [31]. Dependable spanners, which maintain connectivity despite edge failures, further support the reliability of sharded systems [67].

Advanced sharding techniques mirror evolutionary data systems, introducing modular architectures for flexible transitions between storage methods, thus enhancing scalability. The use of materialized views (MVs) to precompute expensive joins significantly boosts performance in NoSQL databases, showcasing sharding's versatility in optimizing query execution [59]. Polyglot data management approaches that partition databases to accommodate multiple data models highlight sharding's role in distributed environments. The effectiveness of sharding is closely tied to constructing spanners for metric spaces that retain properties post-failure of vertices, enhancing resilience and scalability [68]. Leveraging snapshots to maintain operational order simplifies implementation and boosts scalability.

Sharding facilitates concurrent access to large data objects while supporting strong consistency guarantees via mechanisms like fragmented linearizability. Systems such as COBFS utilize sharding to organize files as linked lists of coverable block objects, enhancing concurrency for large objects. Innovations like HopsFS demonstrate sharding's ability to alleviate metadata bottlenecks in hierarchical file systems, yielding improved throughput and capacity compared to traditional systems like HDFS [31, 54]. By integrating spanner construction and fault tolerance concepts, sharding empowers distributed databases to handle increased loads and data volumes efficiently, ensuring robust performance in modern computing environments.

5.2 Spanner Construction and Its Relation to Sharding

The construction of Google Spanner and its sharding implementation are vital for achieving scalability and reliability in distributed databases. Google Spanner employs advanced spanner construction techniques to manage extensive datasets across geographically distributed nodes, ensuring high availability and consistency even amid network partitions and node failures. It utilizes local computation algorithms (LCAs) for dynamic edge inclusion into a sparse spanner without computing the entire structure. The design incorporates reliable spanner principles, ensuring performance and connectivity despite significant node failures [69, 70].

A key aspect of Spanner's architecture is hierarchical partitioning, related to constructing dependable spanners that endure high edge failure rates while minimizing connectivity loss [67]. This method enhances fault tolerance by preserving connectivity and data integrity despite node or network path failures, crucial for Spanner's reliability.

Spanner also utilizes distributed algorithms for spanner construction, such as the Distributed-Spanner algorithm, which achieves efficient spanner construction through localized information [66]. These algorithms streamline communication and coordination among nodes, reducing overhead in maintaining a consistent state across the distributed system. By leveraging such approaches, Google Spanner optimizes data replication and synchronization, contributing to its scalability.

The architecture incorporates techniques akin to Synergy, which combines schema-based and workload-driven materialized views with lightweight concurrency control for optimized data management in NoSQL databases [59]. This synergy allows efficient data management and query execution, enhancing scalability through optimized communication and computation.

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 [68]. This capability enhances the system's handling of 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 connectivity and performance, underscoring Spanner's architecture efficacy in optimization [68]. Organizing net points into incubators for managing connections and maintaining low degree also contributes to robust and efficient spanner construction [71].

5.3 Reliable Spanners and Fault Tolerance

Reliable spanners are crucial for fault tolerance in distributed systems, ensuring connectivity and performance amidst node or edge failures. Their construction involves sophisticated algorithms that balance efficiency and resilience, enabling seamless operations under adverse conditions [72]. A significant challenge in constructing reliable spanners is balancing stretch and maximum degree. Many methods struggle to effectively limit both, but advancements have introduced improved

deterministic distributed spanner constructions that reduce communication rounds and message sizes, enhancing efficiency [73].

Resilient graph spanners focus on improved local resilience to edge failures and more effective maintenance of distance properties compared to traditional fault-tolerant spanners [74]. This enhancement is vital for distributed systems, where maintaining connectivity and performance during failures is paramount. Constructing fault-tolerant spanners that preserve distance properties despite vertex failures ensures reliable operations, providing a framework for maintaining connectivity [67].

Furthermore, reliable spanners requiring fewer edges than previous constructions enhance practicality for real-world applications [71]. These spanners maintain robustness while achieving reduced edge counts, making them more space-efficient. The proposed method achieves spanner sizes comparable to undirected graphs, a significant advantage for directed graphs [68].

In addition to construction, reliable spanners benefit from improved round complexity, facilitating faster spanner construction without compromising edge count [68]. This advancement is critical for distributed systems, where minimizing spanner construction time can significantly enhance overall performance. The proposed reliable spanners are designed to maintain fault tolerance, ensuring distributed systems remain resilient and efficient in dynamic environments.

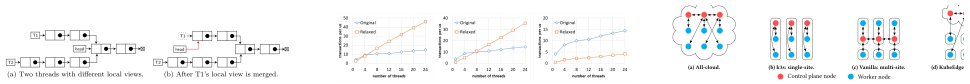
5.4 Advanced Techniques for Scalability

Enhancing scalability in distributed systems necessitates innovative techniques beyond traditional sharding methods. Reliable spanners are pivotal for maintaining performance and connectivity amid significant node failures. Their construction involves two steps: initially constructing spanners for uniform metrics and then utilizing them to address general metrics through covers [68]. This novel approach significantly enhances performance and scalability.

Dynamic partitioning strategies also promise to improve scalability, particularly in systems like HopsFS, where metadata operation hotspots pose challenges. Future research should explore these strategies to further enhance performance and address scalability issues linked to concentrated data access patterns [31].

Integrating locality-sensitive orderings in reliable spanner construction improves stretch and sparsity while withstanding massive node failures. This approach extends to broader metric classes, providing robust solutions for distributed systems facing disruptions [68]. Additionally, developing relaxed spanners for directed disk graphs indicates potential advancements in compact routing schemes and distance oracles, leveraging insights to improve scalability.

Collectively, these advanced techniques enhance distributed system scalability by optimizing communication, computation, and data management while managing trade-offs between consistency and availability. Through centralized and decentralized coordination strategies, these methods enable controlled adjustments to inconsistency and unavailability in response to varying network latency. This nuanced approach allows systems to maintain operational efficiency and reliability despite network partitioning, addressing complexities highlighted by the CAP theorem and its extensions [17, 19]. By transcending traditional sharding methods, these strategies ensure distributed systems can efficiently manage increased loads and complex data structures, sustaining robust performance in dynamic environments.



(a) Two threads with different local views[75]

(b) Performance Comparison of Original and Relaxed Algorithms in Distributed Systems[76]

(c) Understanding Cloud Computing: Different Architectures and Their Applications[35]

Figure 8: Examples of Advanced Techniques for Scalability

As illustrated in Figure 8, scalability is crucial for ensuring efficient and reliable performance in distributed systems and cloud computing as demand increases. The interplay of sharding and scalability is examined through various advanced techniques. The first example, "Two threads with different local views," highlights the complexity of thread interaction with shared memory structures,

emphasizing the significance of local views for scalability in concurrent systems. The second example, "Performance Comparison of Original and Relaxed Algorithms in Distributed Systems," presents an empirical analysis of algorithmic efficiency, contrasting original and relaxed approaches across varying thread counts to demonstrate their impact on transaction throughput. Lastly, "Understanding Cloud Computing: Different Architectures and Their Applications" offers a comparative examination of cloud architectures, such as All-cloud and KubeEdge, emphasizing diverse strategies in network topology and resource allocation that contribute to scalable cloud solutions. Together, these examples underscore the multifaceted strategies employed to enhance scalability in modern distributed systems [75, 76, 35].

6 Consistency Models: Linearizability and Eventual Consistency

Consistency models are fundamental in distributed systems to ensure the correctness of concurrent operations. This section explores linearizability and eventual consistency, two pivotal models. Linearizability ensures operations are atomic and ordered, whereas eventual consistency prioritizes availability and partition tolerance. Understanding these models is essential for balancing strict consistency with efficiency. The following subsection delves into linearizability, elucidating its principles and relevance in concurrent data structures.

6.1 Understanding Linearizability

Linearizability is a key consistency model that ensures operations on shared objects appear instantaneous between invocation and completion, serving as a correctness criterion for concurrent data types [77]. It allows reasoning about concurrent data structures, ensuring they operate as if in a sequential order respecting real-time [4]. This model is crucial for verifying concurrent algorithms, aligning them with sequential specifications to maintain data integrity. Interval-linearizability offers a refined view of operation interactions [4].

Despite its importance, linearizability can be computationally complex, especially in verifying operations with intricate interactions. To address these challenges, compositional techniques have been developed to improve verification efficiency [77]. While linearizability is often too restrictive, particularly in state-machine replication, alternative models like interval linearizability relax constraints while preserving deterministic execution integrity [78, 79, 80].

6.2 Challenges in Proving Linearizability

Proving linearizability in complex distributed environments is challenging due to the dynamic nature of linearization points (LPs), often dependent on runtime conditions. This complexity is compounded by concurrency and persistency issues in non-volatile memory (NVM) semantics [81]. The NP-completeness of proving linearizability necessitates exploring vast execution possibilities [77], especially in systems using randomized algorithms [82]. The CLAM theorem further complicates this by highlighting limitations in satisfying consistency properties [83].

In asynchronous systems, real-time event ordering is crucial for linearizability, but strict synchronization requirements can hinder performance and scalability [84]. Advanced techniques are needed to verify operations like the Contains operation in lazy set objects, which set linearization points for others [85]. Addressing these challenges requires ongoing research to enhance verification techniques for modern distributed environments [79, 86, 87, 80, 88].

6.3 Eventual Consistency and Its Variants

Eventual consistency ensures updates will eventually propagate to all replicas, achieving convergence over time. It is ideal for applications prioritizing availability and partition tolerance over immediate consistency. Almeida et al. expand consistency models to include synchronization-oriented abstractions, essential for understanding eventual consistency nuances [83]. The epistemic perspective frames eventual consistency as a sequential consistency variant, refining system knowledge about operation order as updates propagate [4, 89, 17, 19].

Variants like causal consistency ensure operations maintain causal relationships, improving data management across distributed nodes. Causal consistency allows greater availability and partition

tolerance while respecting update dependencies, supporting efficient applications in complex environments [29, 26, 90, 91, 92]. Staleness-based models refine eventual consistency by bounding lag time or versions, balancing consistency and performance.

6.4 Trade-offs Between Linearizability and Eventual Consistency

Trade-offs between linearizability and eventual consistency are crucial in distributed system design, affecting performance and reliability. Linearizability ensures atomic operations and real-time order, essential for applications requiring strict data integrity, like financial systems [83]. However, it incurs synchronization overhead, increasing latency and reducing throughput [77].

Eventual consistency enhances performance by allowing asynchronous updates, prioritizing availability and partition tolerance. This model suits applications where immediate consistency is unnecessary, offering improved scalability and reduced coordination overhead. However, oversimplified beliefs about strong vs. eventual consistency indicate complex trade-offs [17]. The CALM Theorem clarifies which programs can execute without coordination [1].

A limitation of eventual consistency is ensuring data consistency across systems, leading to potential gaps in availability and integrity. Trade-offs between trusted servers and protocols like VICOS illustrate consistency verification challenges [2]. The Pathway Theorem provides a structured methodology for proving durable linearizability, crucial for addressing these challenges [81].

Innovative approaches like Intermediate Value Linearizability (IVL) relax linearizability for cost-effective implementations while preserving probabilistic error bounds [89]. Analyzing trade-offs between linearizability and write strong-linearizability emphasizes preserving termination properties in randomized algorithms [82].

Ultimately, choosing between linearizability and eventual consistency should consider application requirements, such as the need for immediate consistency, tolerance for temporary inconsistencies, and the desired performance-reliability balance. Understanding these trade-offs allows system designers to make informed decisions, ensuring robust operations in distributed environments [4].

6.5 Innovative Approaches and Future Directions

Innovative approaches to consistency models are evolving to meet modern computing demands. Refining locality-sensitive orderings enhances their applicability across metric spaces, potentially advancing computational geometry and graph theory [72]. This refinement could optimize reliable spanner construction, crucial for connectivity and performance amid node failures [74].

Developing vertex fault-tolerant (VFT) k-spanners, achieving fault tolerance with minimal spanner size increase, represents significant improvement [93]. Future research could explore their application in distributed scenarios, enhancing resilience and efficiency. Further refinements to local linearizability could improve its applicability in complex data structures and integration with other models, offering robust solutions for concurrent programming [84].

In linearizability, the Abstract History Method offers a systematic approach to proving linearizability, simplifying verification for complex algorithms [85]. Future research will extend this method to other data structures and explore automation potential [87]. Exploring deterministic specifications could yield new insights into linearizability complexity [94].

Optimistic execution techniques present another innovation avenue, where future research could integrate monitoring with automated rollback systems and develop efficient predicate detection algorithms [24]. These advancements could lead to efficient verification processes and robust consistency guarantees.

Applying the Pathway Theorem to persistent data structures could optimize flushing strategies and enhance linearizability durability [81]. Exploring these innovative approaches and future directions can enhance distributed systems' reliability and efficiency, addressing modern computing challenges.

Future work could focus on refining existing frameworks to better handle specific distributed systems and exploring additional properties to bolster consistency guarantees [83]. This exploration is crucial for developing adaptable and robust consistency models for contemporary applications.

7 Distributed Consensus and Scalability Challenges

7.1 Challenges in Achieving Distributed Consensus

Achieving distributed consensus in large-scale systems involves navigating complexities related to consistency, coordination, and fault tolerance in dynamic environments. Ensuring linearizability across geo-distributed data centers is crucial, yet challenging, due to latency and operational cost constraints, especially under dynamic workloads and potential failures. Robust synchronization mechanisms are essential for maintaining consistent states across distributed nodes [21]. The verification of linearizability is further complicated by the exponential growth of the search space, reaching sizes of O(N!) for histories of length N [77].

Integrating new data stores disrupts seamless query processing, necessitating consensus mechanisms that efficiently manage diverse environments [9]. The verification of linearizability in concurrent data structures, such as the Lazy Set algorithm, demands a mathematical framework capable of proving correctness amid concurrent operations [85].

In environments with potentially malicious servers, ensuring data integrity and consistency is paramount. Protocols like VICOS enable clients to verify their data's integrity and consistency without fully trusting the server [2]. The inability of current methods to handle concurrent operations without high implementation costs hinders efficient distributed consensus [89].

The threat posed by strong adversaries, who can manipulate execution schedules to prevent termination, underscores the necessity for robust consensus mechanisms capable of withstanding adversarial conditions [82]. Traditional coordination mechanisms often lead to performance bottlenecks, prompting exploration of problems solvable without coordination to enhance efficiency [1].

Moreover, constructing spanners with non-constant lightness bounds limits their practical applicability, indicating a need for more efficient spanner construction techniques to support consensus processes [66]. Addressing these challenges requires ongoing research and innovation to develop scalable distributed consensus mechanisms for complex environments.

7.2 Innovative Algorithms and Techniques for Consensus

Advancements in consensus processes are driven by innovative algorithms and techniques addressing coordination, fault tolerance, and communication efficiency. The Distributed-Spanner algorithm optimizes communication among nodes using localized information, reducing overhead and maintaining efficiency as systems scale [66]. Intermediate Value Linearizability (IVL) enhances traditional linearizability by allowing intermediate return values, facilitating cost-effective implementations and improving consensus efficiency [89].

The P-compositionality based linearizability checker (PCL) partitions operation history into independent subproblems, enhancing efficiency through parallel verification and reducing computational complexity [77]. The development of write strong-linearizability ensures robust guarantees for data consistency and reliability [82]. Additionally, the Lazy Set algorithm's two-layer proof structure enhances verification of concurrent data structures [85].

These innovations signify substantial advancements in distributed consensus, providing robust solutions for achieving agreement in complex environments. By integrating advanced consistency models and structured verification frameworks, these methodologies improve reliability and efficiency, facilitating scalable, resilient distributed systems. They address trade-offs between consistency and availability, as articulated by the CAL theorem, and leverage formal specifications and static analysis techniques to ensure correctness in weakly consistent environments [34, 27, 19].

7.3 Strategies for Overcoming Scalability Challenges

Overcoming scalability challenges in distributed consensus mechanisms demands innovative strategies to enhance performance and efficiency. Developing improved deterministic distributed construction methods for graph spanners optimizes communication efficiency and reduces overhead. Future research should extend these methods to accommodate diverse graph structures and optimize algorithms for specific applications [95].

Integrating advanced consensus protocols, such as DAG-based protocols, enhances transaction throughput and reduces latency by allowing parallel processing of transactions. Future research should focus on developing standardized performance metrics, exploring fairness in transaction handling, and enhancing privacy mechanisms for robust, scalable consensus solutions [96].

High-performance consensus mechanisms, such as Nezha, highlight the importance of optimizing clock synchronization and integrating advanced cloud technologies to enhance performance and reliability [40]. Exploring optimizations for communication efficiency in message-passing models can significantly contribute to overcoming scalability challenges, reducing latency, and improving coordination among nodes [97].

8 Conclusion

The survey of distributed systems and databases underscores several foundational concepts and technologies vital for building resilient and scalable infrastructures. A detailed examination of the CAP theorem, ACID and BASE models, consensus algorithms, and consistency models reveals their crucial role in maintaining reliability and performance in distributed settings. The CAP theorem elucidates the inherent trade-offs between consistency, availability, and partition tolerance, serving as a guide for system architects in decision-making processes. Meanwhile, the ACID and BASE models offer contrasting paradigms for transaction management, impacting how systems balance consistency and availability.

Consensus algorithms, including Paxos and Raft, are integral to achieving consensus across distributed nodes, thus ensuring data consistency and system dependability. The evolution of protocols such as Egalitarian Paxos (EPaxos) marks significant progress in scalability, albeit with certain safety challenges. The transition from NoSQL to NewSQL, exemplified by systems like Google Spanner, highlights advancements in scalability and consistency, aligning with the needs of modern, data-intensive applications.

Sharding and scalability techniques remain pivotal in addressing the complexities inherent in distributed systems. Methods such as hierarchical partitions and spanner constructions improve system efficiency and fault tolerance, with innovations like lightweight bounded-degree spanners exemplifying optimization prowess. Consistency models, ranging from linearizability to eventual consistency, provide varying degrees of data integrity and availability, tailored to specific application requirements. The classification of eventually linearizable objects enhances understanding of their synchronization capabilities and applications.

In light of the evolving landscape of distributed systems and databases, continuous research and innovation are imperative for addressing emerging challenges and enhancing performance. Future endeavors should focus on developing programming languages and tools that facilitate the creation of monotonic distributed systems and exploring the expressive potential of monotonic programs. Furthermore, advancements in polyglot data management frameworks are crucial for adapting to shifting data demands and technological advancements. Mastery of these core concepts is essential for developing distributed systems that are robust, scalable, and adaptive to the dynamic requirements of contemporary computing environments. As the field progresses, the integration of novel algorithms and methodologies will be key to advancing distributed systems and databases, ensuring their efficacy in managing complex data 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] Priyam Shah, Jie Ye, and Xian-He Sun. Survey the storage systems used in hpc and bda ecosystems, 2021.

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

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

[19] Edward A. Lee, Soroush Bateni, Shaokai Lin, Marten Lohstroh, and Christian Menard. Quantifying and generalizing the cap theorem, 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] Matthieu Perrin, Achour Mostéfaoui, and Claude Jard. Brief announcement: Update consistency in partitionable systems, 2015.

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

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

[30] 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.

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

[32] Muntasir Raihan Rahman, Lewis Tseng, Son Nguyen, Indranil Gupta, and Nitin Vaidya. Characterizing and adapting the consistency-latency tradeoff in distributed key-value stores, 2016.

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

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

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

[36] 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.

[37] Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners, 2018.

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

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

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

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

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

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

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

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

[46] Improved deterministic distribut.

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

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

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

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

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

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

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

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

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

[56] 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.

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

[58] 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.

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

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

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

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

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

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

[65] Didier Galmiche, Timo Lang, and David Pym. Minimalistic system modelling: Behaviours, interfaces, and local reasoning, 2024.

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

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

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

[69] Sariel Har-Peled, Manor Mendel, and Dániel Oláh. Reliable spanners for metric spaces, 2022.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[88] Haoxiang Lin. A constructive proof on the compositionality of linearizability, 2018.

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

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

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

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

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

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

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

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

[97] 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.