International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Nested Transaction Model: A Comprehensive Review of Mechanisms, Challenges, Applications, and Future Prospects

Meenu

Submitted:12/03/2024 **Revised**: 27/04/2024 **Accepted**: 04/05/2024

Abstract: This review paper explores the foundational and advanced elements of nested transaction models in database systems, presenting a structured analysis across key areas. Beginning with an introduction to nested transactions, the paper reviews essential concurrency control and commit protocols tailored for nested environments, highlighting their impact on transaction management. Deadlock detection techniques are analysed with a focus on their applicability and efficiency in nested scenarios. The study also examines various recovery models designed to maintain consistency and reliability within nested transactions. Key research issues are identified, pinpointing gaps and limitations in current methodologies. The paper further discusses challenges associated with implementing nested transactions, including performance constraints and scalability issues. Applications of nested transactions within database contexts are presented, alongside broader applications in complex data environments, showcasing the versatility and potential of these models. Finally, future research directions are proposed, emphasizing areas for improvement and innovation, followed by a comprehensive conclusion summarizing insights gained and their implications for advancing nested transaction models in database systems.

Index Terms: *Challenges of Nested Transactions, Commit Protocols, Concurrency Control, Database Applications, Deadlock Detection.*

I.INTRODUCTION

This section traces the journey from manual filing systems to advanced database solutions, focusing on the transition to centralized (CDBS) and distributed database systems (DDBS) that support networked data access. It highlights the emergence of real-time database systems (RTDBS) for applications with strict timing requirements, including Distributed Real-Time Database Systems (DRTDBS). It then categorizes transaction models, noting significant contributions by researchers, and introduces the Advanced Transaction model. The section also distinguishes between closed and open nested transactions, emphasizing the benefits of nested transactions like failure independence, intratransaction parallelism, and modularity. Finally, it outlines the paper's key focus areas, including the progression of concurrency control mechanisms and the unique challenges posed by nested transactions.

The shift from manual filing systems to computerized solutions has marked significant milestones in information management. Early file-based and database systems aimed to improve data management efficiency but faced challenges like the lack of integrated data definitions and limited data control. This evolution laid the groundwork for centralized database systems (CDBS), which enabled users to define, create, maintain, and control database access [1] . With the growth of network technologies, distributed database systems (DDBS) emerged to link databases across computer networks [2] . Recently, real-time database systems (RTDBS) have gained traction, particularly in applications that require time-sensitive data and specific transaction deadlines [3]. Key sectors benefiting from RTDBS include aircraft control, stock trading, network management, and factory automation [4], [5].In real-time applications, transactions are categorized as hard, firm, or soft real-time based on their deadline requirements [6] . This categorization has extended to interconnected databases, leading to the development of Distributed Real-Time Database Systems (DRTDBS) [7] .A fundamental unit in database transactions, represented as $T = << t, Ai, Ni >\cdots i=1\ldots n>$, involves operations such as reading and writing data, consisting of predefined steps [8] .Transaction models are essential in database management systems, providing frameworks to manage and ensure the consistency and reliability of database transactions. Over the years, various transaction models have been proposed, each targeting specific challenges and optimizing data management processes. These models, developed by distinguished researchers and database experts, exemplify the diverse strategies employed to enhance transaction processing efficiency.

The evolution of transaction models has been pivotal in enhancing data management and addressing the complexities of modern database systems. These models have undergone significant transformations to cater to various requirements, from simple operations to complex applications involving real-time constraints. Below is a comprehensive overview of different transaction models

Department of CSE, M. M. M. U. T., Gorakhpur, India
**myself_meenu@yahoo.co.in*

presented in Table I, highlighting their structures, merits, demerits, and challenges. This classification includes both traditional and advanced transaction models, providing insights into their respective functionalities and applicability in database management.

TABLE I

COMPARATIVE ANALYSIS OF VARIOUS TYPES OF TRANSACTION MODELS

S.No.	Type of Transaction Model	Description	Transaction Structure	Merits	Demerits	Challenges
1.	Gray's Model [9], [10]	Efficient data management	Multi-level locking, Two-phase commit	High concurrency, Robust recovery	Complexity, Locking overhead	Data consistency, Distributed transactions
2.	Read/Write Model [11]	Considers database objects as pages in memory	Read/Write operations	Sequences of read and write operations	Ignores user program computations in main memory	Handling complex computations within transactions
3.	Relational Updates [12]	Basic operations: insertions, deletions, modifications	Single tuples	Atomic execution of insertions, deletions, modifications	Applied to relations in a given database	Ensuring consistency in concurrent insertions, deletions, modifications
4.	Online/Batch Transactions [13]	Classifies transactions as online (short life) or batch (long-life)	Individual transactions for online, grouped for batch	Short response times for online, longer for batch	Online: Small portion of the database; Batch: Larger portion	Optimizing resource usage and response times for both online and batch transactions
5.	General/TwoStep/Restricted Two-Step [14], [15], [16]	Based on read and write actions	Defined read and write action sequences	Various based on read/write actions	Various based on read/write restrictions	Managing transactions with specific read and write ordering constraints
6.	Single, Uniform, Distributed DBMS [17].	Distributed DBMS model with four components	Distributed components across sites with consistency control	Study of concurrency control algorithms and performance	Complexity in distributed environments	Developing effective concurrency control algorithms for distributed databases
7.	Flat Transaction Model [18] / Advanced Transaction Model [19]	Classifies transactions as Flat model (Single level)	Flat for Flat model, Hierarchical for	Flat: Improvement over manual file systems;	Flat: Limited to simple transactions; Advanced:	Handling large datasets, Integrity

		or Advanced (multi-level)	Advanced model	Advanced: Suitable for complex applications	Performance overhead	checks; Complexity in design for Advanced model
7.1.	SAGA [20] (Advanced)	Single level of nesting with compensating transactions	Hierarchical	Simpler than nested model, has compensating transactions	Defining compensating transactions in advance	Limited nesting levels, Complex recovery actions
7.2.	Workflow Model [21] (Advanced)	Combines open and nested transactions, with workflow specification	Hierarchical	Application-specific correctness	Workflow specification complexity	Efficient specification and execution of workflows
7.3.	Dynamic Restructuring [22] (Advanced)	Adaptive recovery with split and join-transaction operations	Hierarchical	Adaptive recovery, Reduced isolation	Not suitable for modelling business activities	Handling transaction joins, Complexity in restructuring
7.4.	Flex Transaction Model [23] (Advanced)	Extends ACID properties hierarchically, enhancing flexibility	Hierarchical	Greater flexibility in transaction management	Increased complexity and performance overhead	Balancing flexibility with manageable complexity
7.5.	Nested Model [24] (Advanced)	Allows arbitrary level of nesting with a transaction tree structure	Hierarchical	Failure independence, Intra-transaction parallelism, Modularity	Complexity	Managing nested transactions and increased development time
7.6.	Multilevel Model [25] [26] (Advanced)	Balanced tree of subtransactions, specialization of open nested model	Hierarchical	Balanced tree structure	Not suitable for business activities	Lack of flexibility, Limited scalability

The Table I above captures a range of transaction models, encompassing both foundational and advanced structures. These models collectively illustrate the evolution from simple flat transactions, which suit basic data management, to sophisticated advanced models like Nested and Workflow, which address the complexities of modern database systems. Flat and basic models lay the groundwork by focusing on straightforward transaction execution, while advanced models, such as Nested and Dynamic Restructuring, introduce structures that support failure independence, modularity, and adaptability. These advanced models are critical in environments requiring high concurrency and resilience, especially in real-time and distributed systems. Together, they highlight the ongoing advancements in transaction management, which cater to both traditional needs and the dynamic requirements of complex applications. Further development and research in this field will continue to optimize and expand these models to support future challenges in database technology.

In transaction model development, a variety of approaches aim to achieve efficient data management across complex applications. The transition from the Flat transaction model to more sophisticated, advanced models emerged from the limitations of flat models in addressing engineering complexities, particularly in fields like CAD and software engineering. The Flat model, characterized by a single initiation and termination point, is insufficient for handling intricate, prolonged processes. Advanced models, particularly the Nested transaction model, play a critical role in these scenarios by offering operational abstractions and flexible handling of ACID properties. Moss introduced the Nested transaction model, drawing on the "spheres of control" concept from Bjork and Davies [27], [28].This model structures transactions hierarchically, with a top-level transaction and subtransactions forming a "transaction tree." The N-ACID properties apply primarily to the top-level transaction, while subtransactions maintain a limited subset of these properties [29] .Nested transactions are classified into closed and open types. Closed nested transactions keep subtransactions' effects within the parent's scope, with their commitment dependent on the parent's commitment [20], [30] .In contrast, open nested transactions allow subtransactions to operate and commit independently, releasing leaf-level locks early if operation semantics are known [19] , [31], [32] .Unlike flat models, nested transactions support independent failure and rollback of subtransactions without affecting the parent, known as "failure independence," reducing the need for full transaction rollbacks. They also enable intra-transaction and intertransaction parallelism, enhancing performance and modularity, which in turn provides benefits like improved encapsulation and security. These key advantages—failure independence, intratransaction parallelism, and modularity—make nested transactions particularly suitable for real-time, complex, and distributed applications [33].In addition to Nested transactions, models like Sagas, Multilevel, Dynamic Restructuring, and Workflow Models represent significant advancements, providing more adaptable frameworks for applications with complex transaction needs. This research paper focuses on closed nested transaction models, examining their commit and concurrency control protocols and analysing parallelism types, including parent/child and sibling parallelism [33].

The upcoming sections provide a comprehensive exploration of nested transactions, addressing key concepts and issues integral to their implementation and optimization Section 2 focuses on Concurrency Control and Commit Protocols in Nested Transaction, discussing the development of database concurrency control mechanisms, including various concurrency control protocols, the issue of priority inversion within Two-Phase Locking (2PL), and advancements in commit and concurrency control protocols tailored for nested transaction. Section 3 examines Deadlock Detection in Nested Transaction, exploring types of deadlocks, detection strategies, and the challenges involved in resolving deadlocks in complex systems. The review continues with Section 4, which covers Recovery Models in Nested Transaction, focusing on recovery concepts and examining various recovery algorithms designed to handle hierarchical dependencies and ensure consistency. Section 5 highlights Research Issues in Nested Transaction Model, identifying open questions and areas for further investigation aimed at optimizing system performance. Section 6 addresses the Issues and Challenges of Nested Transaction, analysing difficulties related to scalability, system complexity, and implementation. Section 7 explores Applications of Nested Transaction in Databases, illustrating practical use cases and benefits of nested transactions in modern database systems. Section 8 extends the discussion to Key Applications of Nested Transaction, broadening the scope to include domains beyond databases, such as distributed systems and real-time applications. Section 9 looks toward Future Research Directions, outlining emerging areas of research to enhance the scalability, concurrency, and fault tolerance of nested transaction systems. Finally, Section 10 concludes the review with a summary of the key findings and opportunities for further research.

II.CONCURRENCY CONTROL and COMMIT PROTOCOLS in NESTED TRANSACTION

This section explores the progression of database concurrency control mechanisms, with a particular focus on the unique challenges presented by nested transactions in distributed real-time databases. It underscores the need for advanced protocols to achieve global serializability and optimize real-time performance. The discussion addresses various concurrency control protocols and their vital role in ensuring data consistency, as well as managing issues like deadlocks and resource contention. It also covers priority inversion challenges associated with the Two-Phase Locking (2PL) protocol, tracing the development of concurrency control and commit protocols designed specifically for nested transactions. The review highlights major advancements while acknowledging persistent issues, such as cascading intra-aborts, reinforcing the demand for specialized protocols to enhance transaction processing efficiency in distributed systems.

In database management, enabling concurrent access for multiple users is crucial, as it leads to increased throughput, decreased transaction waiting times, and enhanced overall system performance. While read

operations generally do not cause conflicts, the simultaneous execution of write operations can create complexities, resulting in issues such as lost updates, uncommitted dependencies (dirty reads), and inconsistent analyses. To mitigate these challenges and maintain consistency and reliability in database operations, effective concurrency control mechanisms are essential. Despite significant advancements in real-time concurrency control across various transaction models, the unique challenges posed by nested transactions in distributed real-time databases necessitate specialized protocols. This proposed research seeks to address these challenges by emphasizing global serializability and real-time performance in the context of nested distributed transactions. Building upon a thorough examination of existing literature, this work provides a comprehensive overview of the historical evolution of concurrency control protocols and the development of commit protocols tailored specifically for nested distributed real-time database systems. By innovatively addressing the distinct challenges associated with nested transactions within dynamic distributed environments, this research aims to bridge existing gaps in knowledge. Ultimately, it contributes

valuable insights to the complex field of concurrency control in distributed real-time databases, pushing the boundaries of current understanding and practice.

In conclusion, this research advances the field of concurrency control by tackling the specific challenges of nested transactions in distributed real-time databases. By proposing innovative approaches to improve global serializability and real-time performance, it provides valuable insights and addresses key limitations in existing protocols, thereby filling critical gaps and contributing meaningfully to the evolving landscape of concurrency control.

A. DATABASE CONCURRENCY CONTROL MECHANISMS

This section examines concurrency control protocols critical for ensuring data integrity and consistency in multi-user databases. It categorizes protocols into locking and timestamping, highlighting pessimistic and optimistic approaches. Real-time databases face specific challenges that require priority-based scheduling methods. The discussion emphasizes the importance of choosing the appropriate protocol for effective transaction management as technology advances.

In database management, concurrency control protocols are essential for ensuring transaction integrity and data consistency across concurrent operations. These protocols are broadly categorized into locking, timestamping, and real-time mechanisms, employing various strategies to manage and resolve conflicts. Locking and timestamp-based protocols can utilize either pessimistic or optimistic approaches; pessimistic methods assume a high likelihood of conflict and proactively prevent issues, while optimistic methods operate under the assumption that conflicts are rare, only intervening when necessary. Notably, Two-Phase Locking (2PL) and Timestamp Ordering (TO) protocols offer variations designed for scalability and specific use cases, particularly in distributed systems. In real-time databases, where timing is crucial, priority-based approaches like First Come First Serve (FCFS) and Earliest Deadline First (EDF) optimize performance and predictability. Additionally, concurrency control is a critical aspect of database management systems, ensuring transactions are executed in a controlled and consistent manner within multi-user environments. This involves managing access to shared resources to prevent conflicts and maintain data integrity. A key concept in concurrency control is serializability, which aims to identify schedules where transactions can run concurrently without conflicts [14] thereby ensuring that the database state accurately reflects a sequential execution. The Table II below summarizes key concurrency control protocols, their approaches, descriptions, and practical applications.

TABLE II OVERVIEW OF DATABASE CONCURRENCY CONTROL MECHANISMS

S.NO.	Concurrency Control Protocol	Approaches	Description
1	Locking and Timestamping	Pessimistic Approaches [34].	Pessimistic Techniques: Assuming conflicts are frequent, these approaches—covering lock-based, order-based, and hybrid methods—focus on preventing conflicts proactively. Two-Phase Locking (2PL) is a prominent example of

			pessimistic locking that helps prevent issues like lost updates, uncommitted dependencies, and inconsistent reads.
		Optimistic Approaches [34].	Optimistic Techniques: Based on the assumption that conflicts are rare, optimistic approaches involve reading, validating, and writing data at distinct stages. Techniques such as optimistic locking and timestamp ordering are commonly used to increase concurrency. [35] [36], [37]
1.1	Two-Phase Locking (2PL) [38]		Two-Phase Locking Protocol (2PL): Divided into growing and shrinking phases, 2PL is designed to achieve conflict serializability. To address challenges like cascading rollbacks, variants such as rigorous 2PL and strict 2PL have been developed. In distributed systems, variations such as Primary Site 2PL [39] , Primary Copy 2PL [40], Voting 2PL [36], , and Distributed Two-Phase Locking (D2PL) help reduce data redundancy, with implementations found in systems like System R* [41] and NonStop SQL. [42], [43], [44]
1.2	Timestamp Ordering (TO)	Pessimistic TO	Pessimistic Timestamp Ordering: This method prioritizes transactions by timestamps to manage conflicts. Basic TO ensures conflict serializability [45] , while Conservative TO variants introduce delays to minimize

			transaction restarts and reduce system overhead. Multiversion TO further enhances concurrency by allowing multiple data versions [46], [47] [48], [49], [50] .
		Optimistic TO	Optimistic Timestamp Ordering: This approach defers timestamp assignment until validation, promoting concurrency in low-conflict scenarios. Although effective in such cases, it may result in higher storage overhead and transaction restarts.
2	Real-Time Concurrency Control		Challenges and Priority Schemes in RTDBS: Real-Time Database Systems (RTDBS) face unique challenges, such as restarts, repeated restarts, and chained blocking [51] . Priority-based scheduling, including First-Come-First-Serve (FCFS), Earliest Deadline First (EDF), Minimum Slack Time First (MSTF), and Shortest Job First (SJF), is applied to increase predictability and efficiency in managing transactions [52], [53], [54].

Following the overview of concurrency control protocols presented in the Table II, it is evident that each approach plays a vital role in maintaining data integrity and consistency within database systems. The choice between pessimistic and optimistic protocols significantly impacts how transactions are managed, particularly in multi-user environments where the likelihood of conflicts varies. Pessimistic methods, such as Two-Phase Locking (2PL) and Pessimistic Timestamp Ordering, are designed to prevent conflicts proactively, ensuring a high level of data integrity but often at the cost of reduced concurrency. Conversely, optimistic methods, which include Optimistic Locking and Optimistic Timestamp Ordering, allow for greater concurrency by assuming conflicts are infrequent, but they may introduce overhead when conflicts do occur. Furthermore, in the context of real-time databases, implementing effective concurrency control is crucial for meeting strict timing constraints. Priority schemes like FCFS and EDF enhance predictability and responsiveness, ensuring that high-priority transactions receive timely access to resources. As technology continues to evolve, the development of more sophisticated concurrency control mechanisms will be essential for managing increasingly complex and dynamic database environments, facilitating efficient transaction processing and maintaining the overall reliability of database systems.

In conclusion, the discussed concurrency control protocols are crucial for maintaining data integrity and consistency in multi-user database environments. The choice between pessimistic and optimistic approaches, along with adaptations for real-time and distributed systems, emphasizes the need for tailored solutions. As technology evolves, these protocols will continue to play a key role in ensuring efficient and reliable transaction processing.

B. PRIORITY INVERSION in 2PL (TWO-PHASE LOCKING)

This section discusses priority inversion in Real-Time Database Systems (RTDBS) and its impact on high-priority transactions. It presents two strategies for addressing this issue: the Priority Abort Protocol and the Priority Inheritance Protocol, both aimed at ensuring timely execution of critical transactions.

In Real-Time Database Systems (RTDBS), managing priority inversion within the Two-Phase Locking (2PL) protocol is essential for maintaining performance and meeting deadlines of high-priority transactions. Priority inversion occurs when high-priority transactions are delayed due to locks held by lower-priority transactions, which can lead to missed deadlines and compromised system efficiency [55]. This challenge is typically classified into two types: bounded and unbounded priority inversion [56]. Addressing this issue involves strategies such as the Priority Abort Protocol (2PL-HP) and the Priority Inheritance Protocol.

The Priority Abort Protocol (2PL-HP) addresses priority inversion by aborting lower-priority transactions holding locks needed by high-priority transactions, allowing the latter to proceed without delay [4] While effective in reducing priority inversion, this approach may lead to increased transaction restarts, particularly under high contention scenarios. Alternatively, the Priority Inheritance Protocol offers another solution by temporarily raising the priority of a lower-priority transaction holding a lock to match that of the waiting high-priority transaction. This propagation of priority helps prevent indefinite delays, allowing the lower-priority transaction to complete its critical section and release the lock for the higher-priority transaction [57] .

In conclusion, effectively managing priority inversion within Real-Time Database Systems (RTDBS) is crucial for ensuring that high-priority transactions can meet their stringent deadlines. The Priority Abort Protocol and the Priority Inheritance Protocol provide practical solutions to mitigate the adverse effects of priority inversion, enhancing the reliability and efficiency of transaction processing. By addressing the challenges posed by locks held by lower-priority transactions, these strategies contribute to maintaining optimal system performance and ensuring timely execution of critical operations, which is essential in dynamic and time-sensitive environments.

C. EVOLUTION of CONCURRENCY CONTROL and COMMIT PROTOCOLS in NESTED TRANSACTIONS

This section examines the progression of concurrency control and commit protocols specifically designed for nested transactions. It provides a summary of diverse protocols in Table

III, emphasizing their distinct characteristics, advantages, and associated challenges. The continuous advancement of these protocols reflects a commitment to improving scalability and efficiency in managing nested transactions. Database professionals are encouraged to explore these protocols to optimize performance in their systems.

The field of database management has seen significant advancements in both concurrency

control and commit protocols for nested transactions. Table III offers a concise overview of various protocols, detailing their advantages, disadvantages, and the specific challenges they address. This section will explore these protocols, focusing on their origins and unique features, including innovative strategies for managing concurrency and handling intra-abort cascades. By understanding these developments, database administrators can improve their transaction management practices and enhance overall system efficiency.

TABLE III OVERVIEW OF CONCURRENCY CONTROL AND COMMIT PROTOCOLS FOR NESTED TRANSACTIONS

S.No.	Protocol	Description	Merits/ Demerits	Challenges
1.	2PL-NT Concurrency control protocol [24]	Developed Concurrency Control Algorithm for Nested Transactions by integrating Eswaran's [38]	Designed for nested transactions. May have limitations in scenarios beyond nested transactions.	Ensuring compatibility and scalability in various transaction scenarios.

		two-phase locking mechanism.		
1.	Exclusive Locking Algorithm for Nested Transactions [58] [59]	Extending the multi-granularity algorithm tailored for nested transactions.	Tailored for nested transactions. Potential complexity in managing exclusive locks.	Optimizing performance in scenarios with high contention for resources.
2.	Formalization and Generalized Locking Algorithm [60]	Formal proof of Moss's read/write algorithm, introducing a generalized read-update locking algorithm.	Provides a formal proof for concurrency control algorithm. May have overhead in terms of formality and generality.	Application and adaptation in different database architectures.
3.	Extension of Multi-granularity Algorithms [61]	Extended multi-granularity algorithms for nested transactions.	Enhances multi-granularity approaches for nested transactions. Complexity may increase with the extension.	Ensuring backward compatibility with existing systems.
4.	Serialization Graph Construction for Nested Transactions [62]	Serialization graph construction based on I/O automaton models, contributing to nested transaction systems.	Contributes to nested transaction systems through serialization graph construction. ay have computational overhead in constructing serialization graphs.	Optimizing the construction process for large-scale databases.
5.	Formalization of Concurrency Control Algorithms [63]	Formalizing concurrency control algorithms for open and safe nested transactions, utilizing an I/O approach.	Formalizes concurrency control algorithms for open and safe nested transactions. May introduce additional complexity in implementation	Applying the formalized algorithms in real-world scenarios.
6.	Application of Nested Transactions in KBMSs [64]	Applies nested transactions in knowledge base systems (KBMSs).	Applications of nested transactions in KBMSs. Compatibility challenges with existing KBMS architectures.	Adapting to different knowledge base structures and information models.
7.	Extension of Gifford's basic quorum consensus algorithm for data replication to incorporate nested transactions and transactions aborts [65]	Extends Gifford's algorithm for data replication to include nested transactions.	Incorporates nested transactions and transaction aborts into Gifford's basic quorum consensus algorithm. Complexity in managing nested transactions within a consensus algorithm.	Ensuring fault tolerance and consistency in data replication with nested transactions.

8.	Concurrency Control Algorithm for B-trees within nested transactions [66]	Proposed concurrency control algorithm specifically designed for B-trees within nested transaction models.	Tailored for B-trees within nested transactions. May have limitations when applied to other data structures.	Adapting the algorithm to different types of databases and structures.
9.	Concurrency Control Algorithm utilizing linear hash structures for nested transactions. [67]	Presented a concurrency control algorithm utilizing linear hash structures for nested transactions.	Potential limitations in scalability for large databases.	Optimizing the algorithm for diverse database sizes and access patterns.
10.	Multi-version Timestamp Concurrency Control [46]	Introduces a multi-version timestamp concurrency control algorithm for nested transactions.	Provides support for multi-version concurrency control in nested transactions. May increase storage requirements for maintaining multiple versions.	Managing and optimizing storage space while ensuring data consistency.
11.	2PL-NT-HP concurrency control protocol [68] [69]	Introduces concurrency control protocol for Nested Distributed Real-Time Database Systems	Tailored for nested distributed real-time database systems. Overhead associated with high-priority-based scheme.	Ensuring effective conflict resolution and real-time performance.
12.	S-PROMPT commit protocol [68] [69]	Specifically crafted commit protocol for nested transactions to address intra-abort cascade issues.	Addresses issues with intra-abort cascade in nested transactions using PROMPT [70]. Overhead associated with maintaining before and after images.	Optimizing the protocol to minimize performance impact in real-time databases.

The variety of concurrency control and commit protocols for nested transactions in Table III reflects ongoing efforts to enhance database management in complex scenarios. Each protocol addresses specific challenges and introduces innovative strategies. As developers continue to refine their approaches, the future promises more advanced solutions focused on scalability, compatibility, and efficiency. This overview encourages database professionals to explore these protocols to improve nested transaction management.

In conclusion, the evolution of concurrency control and commit protocols for nested transactions highlights significant advancements in database management. The diverse protocols in Table III showcase strategies to tackle challenges while offering distinct advantages, inviting professionals to leverage these innovations for better performance in nested transaction management.

In conclusion, this research highlights the necessity for enhanced concurrency control and commit protocols in nested transactions within distributed real-time databases. By addressing critical challenges such as global serializability, priority inversion, and cascading intra-aborts, the proposed solutions aim to optimize performance and resource management for timely execution of high-priority transactions. This work not only fills significant gaps in existing literature but also sets the stage for future advancements in the field,

ensuring that the complexities of modern applications are effectively managed.

III.DEADLOCK DETECTION in NESTED TRANSACTION

This Section explores deadlock detection in nested transactions, highlighting the complexities due to hierarchical dependencies. It covers types of deadlocks, strategies for detection, and challenges such as resource contention and performance impact. Solutions and algorithmic approaches are discussed to optimize detection and maintain system performance.

In modern transactional systems, managing concurrency and ensuring data consistency are essential but challenging, particularly in environments with nested transactions. In database systems, transactions require locks on data objects to maintain consistency by preventing issues from concurrent access. However, this locking mechanism can lead to deadlocks, where a cycle of transactions waits indefinitely on one another (e.g., T1 → T2 → ... → T1) [35] [71]. Detecting and resolving these deadlocks is critical. One basic approach is the timeout method, where a transaction is aborted if it waits too long for a lock, assuming a deadlock [72]. Although simple, this approach is often imprecise, leading to unnecessary transaction aborts and restarts. A more accurate approach is the waits-for graph (WFG) method, in which the system maintains a directed graph of transactions waiting on each other [73]. Here, each node represents a transaction, and each edge represents a waiting relationship. Deadlocks are identified by detecting cycles within this graph, and the system resolves them by aborting one transaction in the cycle, rolling back its effects, and restarting it. This method is effective for all transaction types and durations, making it a robust deadlock detection solution. Given the rarity of deadlocks, WFG cycle detection is triggered only when needed, which helps optimize performance [74]. However, deadlock detection becomes more complex in nested transactions, where transactions can contain multiple levels of subtransactions. In these cases, deadlocks can occur both across different transaction levels and within the same hierarchy of subtransactions. Unlike single-level transactions, which depend solely on direct waits-for-lock relations, nested transactions require tracking both waits-for-lock and waits-for-commit relationships to fully capture potential deadlock scenarios. This added complexity increases the cost and difficulty of deadlock detection in nested environments.

A. TYPES of DEADLOCKS in NESTED TRANSACTION

This section explores deadlock detection in nested transactions, emphasizing the adaptation of single-level transaction concepts to address hierarchical complexities [75]. It identifies two main types of deadlocks: direct-wait and ancestor-descendant, utilizing the Wait-For Graph (WFG) for cycle detection.

Effective deadlock detection requires extending single-level transaction principles and incorporating additional mechanisms.

1) DIRECT-WAIT DEADLOCKS**:** Occur when a transaction waits for a lock held by another transaction, detectable through direct-waits-for-lock relations in the WFG. A cycle indicates a deadlock, exemplified by mutual waiting between transactions A and B.

2) ANCESTOR-DESCENDANT DEADLOCKS: Arise when a transaction waits for a lock held by its ancestor, affecting the entire hierarchy. Detection involves both direct-waits-for-lock and waits-for-commit relations, ensuring all superior transactions remain in a waiting state until resolution.

In conclusion, the section underscores the importance of effective deadlock detection in nested transactions for maintaining system integrity and performance, highlighting the mechanisms that enable comprehensive detection and management of both direct-wait and ancestor-descendant deadlocks.

B. DEADLOCK DETECTION STRATEGIES in NESTED TRANSACTION

This section examines various deadlock detection strategies in nested transactions, emphasizing the challenges posed by hierarchical dependencies. It compares different approaches, showcasing their unique strengths and limitations. The selection of a strategy must align with the specific requirements of the system, balancing detection accuracy with resource efficiency.

Deadlock detection in nested transactions (NTs) involves unique challenges due to the hierarchical nature of transaction dependencies. In NT systems that utilize locks for concurrency control, committed transactions retain their locks to their parent transactions instead of releasing them. This necessitates that deadlock detection algorithms consider these nested relationships. Numerous algorithms have been developed to tackle these challenges, each presenting distinct advantages and disadvantages.

The Table IV below summarizes a comparative analysis of prominent deadlock detection strategies in NT systems, including Moss's, Rukoz's, Shin's, and Rezende's approaches, highlighting their key features, benefits, and drawbacks. Each deadlock detection strategy in nested transactions has its own strengths and weaknesses. Moss's approach effectively manages nested relationships but may lead to performance overhead. Rukoz's distributed method fits well with nested transaction structures but requires strong communication. Shin's edge-chasing algorithm avoids tree traversal but risks phantom

deadlocks, while Rezende's detection arcs enhance performance but need careful management for accuracy. Choosing a strategy should align with the specific needs of the nested transaction system, balancing effective detection with efficient resource use.

TABLE IV

OVERVIEW OF DEADLOCK DETECTION STRATEGIES FOR NESTED TRANSACTIONS

Deadlock Detection Strategy	**Advantages**	**Disadvantages**
Analyses edges in the wait-for graph to find cycles, managing nested relationships. [76]	Effectively handles nested transactions.	Can incur performance overhead from extensive graph traversal.
Uses a distributed representative graph for hierarchical deadlock detection. [77]	Aligns with nested structures and reduces detection steps after root failures.	Requires robust communication and relies on the root process's reliability.
Implements an edge-chasing algorithm for indirect waiting in parallel nested transactions. [78]	Avoids tree traversal and maintains consistent message overhead.	Prone to phantom deadlocks due to communication delays.
Introduces detection arcs to enhance deadlock management in the wait-for graph. [79]	Improves performance by focusing on fewer graph edges.	Needs careful management to ensure accurate deadlock detection.

In conclusion, analysing deadlock detection strategies highlights their unique attributes and challenges. Each method—Moss's, Rukoz's, Shin's, and Rezende's—offers distinct advantages, emphasizing the need for careful selection based on system requirements to achieve a balance between detection accuracy and resource efficiency.

C. ISSUES AND CHALLENGES in DEADLOCK DETECTION in NESTED TRANSACTIONS

This section highlights the complexities associated with deadlock detection in nested transactions, emphasizing the need for advanced strategies to effectively manage dependencies, dynamic behaviours, and resource contention. These factors are crucial for ensuring reliable performance in transactional systems.

Detecting deadlocks in nested transactions presents several challenges arising from the intricate nature of transaction dependencies, fluctuating transaction behaviours, and competition for resources. Below is a comprehensive analysis of the primary issues related to deadlock detection in nested transactions, along with potential solutions for each challenge. A summary of these issues and their resolutions is provided in Table V.

TABLE V

DEADLOCK DETECTION IN NESTED TRANSACTIONS ISSUES AND REMEDIES

S.No.	Issues in Deadlock Detection in Nested Transactions	Description	Resolution
1	Hierarchical Dependencies	Nested transactions create a hierarchical structure that complicates the modelling and analysis of dependencies.	Implement advanced modelling techniques to accurately represent dependencies within these hierarchies.
2	Cyclic Dependencies	Deadlocks occur due to cyclic waits, where transactions depend on each other for resources. These cycles can extend	Develop algorithms capable of identifying and resolving cycles that span various levels of nesting.

		across multiple nesting levels, complicating detection efforts.	
3	Dynamic Transaction Behaviour	Transactions in concurrent environments display dynamic behaviours, with shifts in states and resource allocations, complicating deadlock detection and resolution.	Utilize adaptive algorithms that can adjust to real-time changes in transaction states and resource allocations.
4	Resource Contention	Resource contention is a key issue in deadlock scenarios, as transactions vie for shared resources, potentially leading to starvation and deadlocks.	Enhance resource allocation strategies to reduce contention and avert cyclic waits.
5	Performance Impact	Mechanisms for detecting and resolving deadlocks can impose computational overhead, adversely affecting system performance. Balancing accurate detection with minimal performance impact is crucial, particularly in high-throughput environments.	Create efficient detection mechanisms that optimize for both accuracy and performance.
6	Transaction Rollback and Recovery	Deadlock resolution frequently requires rolling back transactions and managing recovery processes, which can be complex in nested transactions due to the need for consistency across multiple levels.	Design robust rollback and recovery protocols to maintain data consistency throughout all levels of nesting.
7	Scalability and Complexity	The increasing number of transactions and nesting levels raises the complexity and scalability challenges for deadlock detection algorithms.	Develop scalable algorithms that can effectively manage a large volume of transactions and extensive nesting.

Addressing the challenges of deadlock detection in nested transactions requires advanced algorithms tailored for these complex environments. Effective strategies for resource management, transaction coordination, and deadlock resolution are essential for creating reliable concurrent systems. The resolutions in Table V highlight the need for advanced modelling techniques, adaptive algorithms, and efficient resource allocation methods to effectively manage deadlocks. Implementing real-time detection and resolution systems will help maintain high performance and reliability under high-throughput conditions.

In conclusion, successful deadlock detection in nested transactions is crucial for system reliability and performance. By focusing on advanced strategies and optimizing resource allocation, we can improve the efficiency of detection and resolution, resulting in robust concurrent systems that thrive in demanding environments.

In conclusion, deadlock detection in nested transactions is crucial for maintaining system integrity and performance. The hierarchical dependencies and dynamic behaviours in nested transactions require advanced detection strategies that balance accuracy and efficiency. Challenges such as cyclic dependencies, resource contention, and performance impact must be addressed with scalable algorithms and effective resource management. By leveraging adaptive techniques and robust rollback protocols, systems can efficiently detect and resolve deadlocks, ensuring reliability in complex transactional environments.

IV.RECOVERY MODELS IN NESTED TRANSACTION

This section introduces the concept of recovery in nested transactions, highlighting the importance of restoring databases to a consistent state after a failure. It also explores various recovery algorithms tailored for nested transactions, each offering distinct methods for addressing hierarchical dependencies and ensuring consistency across complex transaction environments.

A. RECOVERY IN NESTED TRANSACTION

This section provides an overview of recovery in nested transaction systems, focusing on the need for specialized methods to manage hierarchical dependencies and ensure data integrity.

Recovery in transaction systems refers to the process of restoring a database to a consistent state after an error, crash, or failure. In transaction processing, recovery mechanisms are critical to maintaining the integrity and reliability of data by ensuring that incomplete or faulty transactions do not leave the database in an inconsistent state. When a failure occurs—whether due to system crashes, power outages, or software malfunctions—recovery protocols work to either commit successfully completed transactions or roll back incomplete ones, preserving data consistency and protecting against data loss. In traditional flat transactions, recovery focuses on handling individual transactions, either completing them or undoing changes to return the system to a previous consistent state. However, as systems have evolved to support more complex applications, such as database management in large-scale, distributed, or real-time systems, the need for more advanced recovery models has grown. Nested transactions, which allow transactions to be structured hierarchically, require specialized recovery models to handle the dependencies between parent transactions and their subtransactions. This hierarchical structure introduces complexities that flat transaction recovery models cannot address, such as coordinating rollback operations across multiple levels and ensuring the consistency of interdependent transactions. Recovery techniques originally developed for flat transactions have been extended to support the unique requirements of nested transactions, where transactions are structured hierarchically with multiple levels [80]. In nested transaction systems, recovery is not limited to restoring individual transactions; it must also consider dependencies and hierarchical relationships between parent and subtransactions.

B. TYPES of RECOVERY ALGORITHMS for NESTED TRANSACTIONS

This section explores various recovery algorithms developed to address the unique requirements of nested transactions, focusing on their ability to maintain data consistency and integrity in hierarchical and complex environments.

Various recovery algorithms have been adapted to this model, each with specific approaches to managing operations and ensuring consistency across transaction levels.

1) INTENTION LIST AND UNDO-LOGGING RECOVERY ALGORITHMS: These two algorithms, initially designed for flat transactions, have been generalized to handle nested transactions by taking advantage of the commutative properties of operations [60] [62] [81]. Both methods focus on the semantics of operations at the leaf level, addressing recovery primarily for individual, lowest level subtransactions. The Intention List records planned changes without immediately applying them, allowing the system to revert to a consistent state if necessary. Undo-Logging records changes before they are executed, enabling rollback by reversing these logged changes. While effective at the leaf level, these algorithms do not fully capture dependencies between higher-level transactions.

2) SYSTEM R MODEL FOR NESTED TRANSACTIONS: The System R recovery model expands upon flat transaction approaches by incorporating layer-specific semantics for two levels of nesting [10]. Unlike the Intention List and Undo-Logging algorithms, which only consider the lowest level of transactions, the System R model accounts for the specific needs of each hierarchical layer. Different recovery rules apply to both parent and subtransaction levels, providing finer control over recovery actions and better handling dependencies. System R's layered approach thus offers a robust and flexible framework for recovery in nested transactions.

3) MULTI-LEVEL TRANSACTION RECOVERY MODEL: Supporting inter-transaction recovery, this model addresses dependencies and interactions across the transaction hierarchy, which is especially beneficial in nested transactions where subtransactions may span multiple levels [25] [82]. By implementing recovery

mechanisms across the entire structure, the multi-level model ensures consistent restoration not only of individual subtransactions but also of their relationships with other transactions, reducing the risk of cascading failures and preserving the nested structure's integrity.

4) WRITE-AHEAD LOGGING (WAL): The Write-Ahead Logging (WAL) algorithm is a crash recovery method adapted for nested transactions, ensuring that any changes are logged before they are applied to the database [83] [84]. This enables recovery by replaying the log to restore a consistent state after a crash. In nested transactions, WAL provides a systematic approach to record changes across all levels, allowing reliable data recovery even in complex, multi-level environments.

Each of these recovery models enhances flat transaction techniques to address nested transactions' unique requirements. Through adaptations for hierarchical structures and inter-transaction dependencies, these models support robust, reliable recovery processes that maintain data consistency and system integrity in complex transaction environments.

In conclusion, effective recovery models are essential for maintaining data integrity in nested transaction systems, especially in the event of failures. By adapting traditional methods such as Intention List, Undo-Logging, System R, Multi-level Transaction Recovery, and Write-Ahead Logging, these advanced models meet the specific needs of hierarchical transactions. Together, they provide reliable mechanisms for restoring data consistency and system integrity across complex, multi-level environments.

V.RESEARCH ISSUES in NESTED TRANSACTION

This section addresses key challenges in real-time nested transactions, focusing on deadline propagation to align subtransactions deadlines and conflict resolution to prevent lower-priority transactions from blocking higher-priority ones, ensuring timely execution in real-time systems.

Real-time nested transaction models are a structured approach in time-sensitive systems, organizing complex tasks hierarchically where a main transaction (or parent) governs multiple dependent subtransactions. These models are crucial for applications requiring precise timing, with parent transactions overseeing multiple subtransactions. A major challenge in this context is ensuring that both parent and subtransactions adhere to deadlines while maintaining concurrency and consistency. The primary research issues in these models centre around deadline propagation and conflict resolution [18]. Deadline propagation involves assigning appropriate deadlines to subtransactions through methods such as absolute, normal, and average propagation. In absolute propagation, all transactions within a family share the same deadline; normal propagation adjusts deadlines based on the relationship between parent and child; and average propagation assigns deadlines based on the average deadlines of subtransactions. Conflict resolution focuses on addressing the Priority Inversion Problem, which arises when a lower-priority transaction blocks a higher-priority one, thereby disrupting the intended priority order. This issue is particularly pronounced in nested transactions, where parent and child transactions may run concurrently. In such cases, a high-priority parent transaction might be blocked by its lower-priority child, causing delays. Techniques like Priority Inheritance and Priority Abort Protocol help to resolve this. Priority Inheritance temporarily raises the priority of a lower-priority transaction to match that of its parent, ensuring that the parent's priority is respected [57]. Alternatively, the Priority Abort Protocol handles conflicts by aborting lower-priority transactions to maintain the correct priority order [4]. These strategies aim to ensure that higher-priority transactions execute first, minimizing delays and preserving the correct sequence of transactions in real-time systems.

In conclusion, real-time nested transaction models effectively manage complex tasks in time-sensitive applications by addressing key issues like deadline propagation and conflict resolution. Methods such as absolute, normal, and average deadline propagation ensure aligned deadlines for subtransactions, while Priority Inheritance and Priority Abort Protocol tackle the Priority Inversion Problem. These strategies work together to support timely, priority-driven execution, enhancing consistency and concurrency in real-time systems.

VI.ISSUES and CHALLENGES of NESTED TRANSACTION

This section explores the challenges posed by nested transactions in database management and presents targeted solutions to address these complexities. It underscores the importance of continued research to optimize performance and uphold data integrity in sophisticated transactional environments.

The implementation of nested transactions brings forth a unique array of challenges, requiring innovative solutions for effective database management. Table VI below provides a detailed examination of specific issues encountered in the nested transaction model, alongside descriptions and proposed resolutions. It explores how developers and researchers have approached complexities such as managing intra-transaction parallelism, establishing prioritization policies for subtransactions, and detecting deadlocks within nested transaction structures. Each challenge is matched with

targeted solutions, including advanced concurrency control mechanisms and specialized protocols that uphold database coherence and ensure global serializability. This overview illuminates the intricacies of nested transaction management, offering valuable insights

into ongoing efforts to enhance performance, maintain data consistency, and address the inherent complexities of layered transactional models.

TABLE VI

NESTED TRANSACTION ISSUES AND REMEDIES

S. No	Issues in Nested Transaction Model	Description	Issues Resolution
1.	Handling of intra-transactions parallelism	In nested transactions, both intra-transaction parallelism and inter-transaction parallelism are present.	To address this issue, implement the following solutions: Concurrency Control Mechanisms, Isolation Levels, Transaction Scheduling Policies, Resource Management, Performance Monitoring, and Tuning, Documentation and Best Practices.
2.	Priority assignment policy for subtransactions [29]	Data sharing among subtransactions can cause delays, so prioritization is needed to avoid execution delays.	To address this issue, implement a priority assignment policy for subtransactions to ensure efficient execution and avoid delays.
3.	Detecting deadlock in nested transactions [70]	Nested transactions require both waits-for-lock and waits-for-commit relations for deadlock detection.	To address this issue, extend deadlock detection mechanisms to consider both waits-for-lock and waits-for-commit relations in the context of nested transactions.
4.	Priority assignment policy [52] [53]	Traditional protocols for RTDBS often neglect transaction priorities.	To address this issue, develop and incorporate priority assignment policies in protocols for real-time database systems to consider transaction priorities.
5.	Concurrency control protocol for subtransactions [68], [69]	Existing concurrency control protocols designed for flat transactions may lead to	To address this issue, design concurrency control protocols specifically tailored for the concurrency of parent and child transactions in nested transaction models.

		issues when applied in a nested environment.	
6.	Commit protocol for subtransactions [49] [69]	Existing commit protocols may face challenges when applied to concurrent parent and child transactions in nested models.	To address this issue, develop commit protocols that address the unique challenges presented by concurrent execution of parent and child transactions in nested environments.
7.	Handling of priority inversion [55]	Priority Inheritance and Priority Abort Protocols are used for priority inversion in advanced transaction models.	To address this issue, implement Priority Inheritance and Priority Abort Protocols to address priority inversion issues in nested transaction models.
8.	Preserving database coherence [85]	Nested transactions are not atomic; thus, the definition of "N-ACID" properties (nested-all-or-nothing, nested consistency, nested isolation, and nested durability) is necessary.	To address this issue, define and ensure N-ACID properties (N-A, N-C, N-I, N-D) for nested transactions to preserve database coherence.
9.	Adjusting transaction recovery according to control structure [79]	Nested transactions have more splitable execution modules and finer control for recovery and concurrency compared to flat transactions.	To address this issue, adapt transaction recovery mechanisms to account for the specific control structure and finer granularity of recovery in nested transactions.
10.	Insurance of global	Nested transaction models have	To address this issue, research and develop methods to ensure the global serializability of distributed real-time nested transactions in real-time database systems.

	serializability [24]	not been fully applied to real-time database systems, raising concerns about the global serializability of distributed real-time nested transactions.	
11.	Handling of transaction parameters in nested transaction [32].	The number of leaves and levels in nested transactions can impact performance.	To address this issue, consider and optimize transaction parameters, such as the number of leaves and levels, to enhance the system's performance in nested transaction models.

Table VI above encapsulates the varied challenges inherent in nested transaction models and presents practical solutions across key areas of database management. Navigating the complexities of nested transactions highlights the need for tailored approaches to ensure smooth execution and preserve data integrity. This examination emphasizes the ongoing efforts in research and development, showcasing the dynamic evolution of database systems. By refining protocols, addressing deadlock scenarios, and implementing priority assignment policies, the database community is unlocking the full potential of nested transactions. As technology advances, so too will our capacity to address these complexities, paving the way for resilient and efficient database management within intricate transactional environments.

In conclusion, this exploration of challenges within nested transactions underscores the complexities of managing such systems. By pinpointing critical issues and suggesting targeted solutions, this section highlights the need for continued research and development to improve performance and uphold data integrity. As the field advances, the database community must persist in innovating and refining protocols to effectively handle the intricacies of nested transactions, setting the stage for more resilient and efficient database management in complex environments.

VII. APPLICATIONS of NESTED TRANSACTION in DATABASES

This section highlights the importance of nested transactions in databases, focusing on their role in error isolation, concurrency, and fault tolerance. It outlines key advancements and their impact on database management systems.

Nested transactions are foundational for managing complex operations in object-oriented and distributed databases. By structuring transactions into subcomponents, they enhance error isolation, support concurrent operations, and improve fault tolerance. This structured approach is essential for maintaining data consistency and reliability across distributed environments, making nested transactions integral to advanced database management systems. The following sections explore key advancements and applications in this area. Early work introduced a generalization of classical serializability theory, aimed at handling complex object bases with nested structures. Following this, a locking protocol was developed to address the commutativity of higher-level methods, allowing conflicts between lower-level methods to be ignored when applicable. Another concurrency control protocol was introduced to provide uniform handling of both class and instance objects. Semantic-based locking protocols were also implemented to enhance traditional nested transaction protocols by incorporating the rich semantics available in object-oriented databases. An open nested transaction model was later applied to mobile databases, organizing mobile transactions as sets of subtransactions. Subsequently, the multi-level transaction model was adapted to maintain replicated data and materialized views by parallelizing updates, and nested transactions were utilized to define workflow processes in hierarchical transaction management environments, streamlining complex transactions. These advancements highlight the essential role of nested transactions in achieving robust and efficient database transaction management.

In conclusion, nested transactions are vital for robust database management, enabling fault tolerance, concurrency, and data integrity in complex, distributed environments. Their evolution through various protocols

and models has strengthened their role in maintaining efficient and resilient transaction processing across modern systems.

VIII.KEY APPLICATIONS of NESTED TRANSACTION

This section explores the key applications of nested transactions in distributed systems, focusing on their ability to enhance fault tolerance by isolating errors within individual components. This isolation allows systems to handle partial failures smoothly, maintaining overall stability and reliability. Through these applications, nested transactions demonstrate their effectiveness in supporting consistency, recovery, and resilience, making them essential for robust distributed computing environments.

Nested transactions are essential for enhancing fault tolerance and recovery within distributed systems by isolating failures within specific components, thus maintaining overall stability and reliability. This structured approach allows systems to handle partial failures in a way that prevents disruptions to entire transactions. Key applications leveraging nested transactions demonstrate these benefits across various distributed environments. For instance, the Argus System [86], [87] applies nested transactions to manage distributed computations with built-in error isolation while the System R [10]Database ensures data consistency through localized recovery mechanisms. Similarly, the Camelot Facility offers robust distributed transaction management, enhancing reliability [88]. The Clouds Operating System [89]increases system resilience by containing local failures, and the Eden File System [90]maintains distributed file consistency along with error isolation. Additionally, the Locus Operating System [91]supports hierarchical failure recovery, and Encina [92]provides atomic, reliable operations for distributed applications. Collectively, these applications underscore how nested transactions strengthen fault tolerance, stability, and reliability, making them invaluable for modern distributed computing systems.

In conclusion, nested transactions enhance fault tolerance, consistency, and reliability in distributed systems by isolating errors within specific subcomponents. Their application across databases and operating systems demonstrates their role in ensuring robust, seamless operations, making them essential for resilient, distributed computing environments.

IX.FUTURE RESEARCH DIRECTIONS

This section discusses future research directions in nested transaction models, highlighting areas for improvement. It also explores the expanding applications of nested transactions, with a focus on enhancing transactional integrity in diverse and dynamic environments.

Future research on nested transaction models can build upon the foundations discussed in this review by exploring several critical areas. In concurrency control and commit protocols, there is an opportunity to develop more robust solutions that address the specific needs of distributed and real-time systems, enabling smoother transaction execution in high-concurrency environments. Enhanced deadlock detection mechanisms are also essential; these methods should focus on minimizing detection time and resource consumption, particularly as transaction complexity grows with deeper nesting levels. Improvements in recovery models for nested transactions can ensure data consistency and system reliability, especially under failure conditions, while handling the unique dependencies between parent and subtransactions. In the realm of research issues, addressing scalability, flexibility, and performance bottlenecks in nested transaction models remains a priority. Issues and challenges related to interoperability, fault tolerance, and resource management also offer significant potential for research advancements, particularly in handling nested transactions in diverse and distributed database architectures. The applications of nested transactions in databases present further avenues for study, with opportunities to enhance database resilience and efficiency across a range of applications, from high-frequency trading to global supply chains. Lastly, the expansion of key applications of nested transactions beyond databases, including fields like IoT, blockchain, and complex event processing, suggests new frontiers for applying these models to maintain transactional integrity in dynamic, multi-system environments.

X.CONCLUSION

This section provides a comprehensive summary of the review paper, covering various aspects of nested transactions and highlighting key insights for advancing research in this area.

In conclusion, this review paper provides a thorough analysis of nested transactions, beginning with Concurrency Control and Commit Protocols in Nested Transactions, which are critical for ensuring consistency and coordination across transaction hierarchies. The discussion on Deadlock Detection in Nested Transactions highlights specific methods for addressing complex inter-transaction dependencies and preventing deadlock situations. We examined Recovery Models in Nested Transactions, focusing on specialized recovery mechanisms that maintain data integrity by addressing the unique hierarchical structures of nested transactions. The section on Research Issues in Nested Transaction Model outlined open questions that remain to be addressed to further optimize transactional performance and reliability. Additionally, we analysed the Issues and Challenges of Nested Transactions, recognizing the complexities and potential limitations in implementing nested transaction models. The review of Nested Transaction Applications in Databases demonstrated the

practical value of nested transactions in managing complex operations, while Key Applications of Nested Transactions showed their adaptability and utility in diverse fields, including distributed and real-time environments. Lastly, Future Research Directions were discussed, identifying opportunities to enhance scalability, concurrency, and fault tolerance to meet the demands of increasingly complex systems. These insights offer a comprehensive foundation for ongoing research and development in nested transaction management.

References

[1] T. C. a. C. E. Begg, Database Systems: A Practical Approach to Design, Implementation and Management, II, Ed., Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[2] M. T. Ö. a. P. Valduriez, Principles of Distributed Database Systems, IV, Ed., Springer, 2020, pp. pp. 1-674.

[3] K. Ramamritham, "Real-time databases," Distributed and Parallel Databases, vol. 01, no. 02, pp. 199-226, 1993.

[4] R. A. a. H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," in 34th International Conference on Very Large Data Bases, 1988.

[5] J. S. a. W. Zhao, "On real-time transactions," in ACM SIGMOD , March 1988..

[6] M. C. a. M. L. J. Haritsa, "Data Access Scheduling in Firm Real-Time Database Systems," International Journal of Real-Time Systems, vol. 4, no. 3, 1992.

[7] M. M. a. A. K. S. U. Shanker, "Distributed real-time database systems: Background and literature review," International Journal of Distributed and Parallel Databases, vol. 23, no. 2, p. 127–149, 2008.

[8] J. Gray, "A Transaction Model," in ICALP, 1980.

[9] J. Gray, "Notes on database operating systems," in Lecture Notes in Computer Science,Operating Systems -- An Advanced Course, vol. 60, Berlin, Springer-Verlag, 1978, pp. 393-481.

[10] J. Gray, "The recovery manager of the system R database manager," ACM Computing Surveys, vol. 13, p. 223–244 , 1981.

[11] S. M. Y. B. H. K. a. A. S. R. Rastogi, "On correctness of non-serializable executions," in 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1993.

[12] S. A. a. V. Vianu, "Equivalence and optimization of relational transactions," Journal of the ACM, vol. 35, pp. 70-120, 1988.

[13] J. Gray, "Why do computers stop and what can be done about it," in CIPS (Canadian Information Processing Society) Edmonton '87 Conference Tutorial Notes, Edmonton, Canada, 1987.

[14] H. Papadimitriou, "Serializability of concurrent database updates," Journal of the ACM, vol. 26, no. 4, p. 631–653, 1979.

[15] P. M. L. I. a. D. J. R. R. E. Stearns, "Concurrency controls for database systems," in 17th Symposium on Foundations of Computer Science, 1976.

[16] H. T. K. a. C. H. Papadimitriou, "An optimality theory of concurrency control for databases," in ACM SIGMOD International Conference on Management of Data, 1979.

[17] M. J. C. a. M. Livny, "Distributed Concurrency Control Performance: A Study of Algorithm, Distribution, and Replication," in 14th VLDB Conference, Los Angeles, California, 1988.

[18] Y. C. a. L. Gruenwald, "Research Issues for a Real-Time Nested Transaction Model," in 2nd IEEE Workshop on Real-Time Applications, July 1994.

[19] M. O. A. E. B. Medjahed, "Generalization of ACID Properties," in Encyclopedia of Database Systems, Boston, MA, 2009.

[20] H. G.-M. a. K. Salem, "Sagas," in ACM SIGMOD International Conference on Management of Data, 1987.

[21] M. R. a. A. Sheth, Specification and execution of transactional workflows, K. W, Ed., ACM Press/Addison-Wesley, 1995, p. 592–620.

[22] Pu, "Superdatabases for composition of heterogeneous databases," in 4th International Conference on Data Engineering, 1988.

[23] Z. a. B. Bhargava, Flex Transactions, Ö. M. Liu L., Ed., Boston, MA: Springer, 2009.

[24] J. E. B. Moss, "Nested Transactions: An Approach to Reliable Distributed Computing," Cambridge, MA, , 1981.

[25] G. Weikum, " Principles and realization strategies of multi-level transaction management," ACM Trans. Database System, vol. 16, no. 1, p. 132–180, 1991.

[26] G. W. a. H. Schek, "Multi-level transactions and open nested transactions," 1991.

[27] L. A. Bjork, "Recovery scenario for a DB/DC system," in ACM Annual Conference, 1973.

[28] T. Davies, "Recovery semantics for a DB/DC system," in ACM Annual Conference, 1973.

[29] R. Guerraoui, "Nested transaction: Reviewing the coherence contract," Elsevier Sciences Journal, vol. 84, p. 161–172, 1995.

[30] G. Karabatis, "Nested Transaction Models," in Encyclopedia of Database Systems, New York, 2017.

[31] Buchmann, "Open Nested Transaction Models," in Encyclopedia of Database Systems, New York, 2016.

[32] H. S. H. a. M. E. E.-S. A. A. EI-Sayed, "Effect of shaping characteristics on the performance of nested transactions," Information and Software Technology, vol. 43, no. 10, pp. 579-590, 2001.

[33] T. H. a. K. Rothermel, “Concurrency Control Issue in Nested Transactions,” VLDB J, vol. 2, no. 1, pp. 39-74, 1993.

[34] P. A. B. a. N. Goodman, “Concurrency control in distributed database systems,” ACM Computing Surveys, vol. 13, no. 2, p. 185–222, 1981.

[35] V. H. a. N. G. P. A. Bernstein, Concurrency Control and Recovery in Database Systems, Addison Wesley, 1987.

[36] R. H. Thomas, “A majority consensus approach to concurrency control for multiple copy databases,” ACM Transactions on Database Systems, 1979.

[37] H. T. K. a. J. T. Robinson, “On optimistic methods for concurrency control,” ACM Transactions on Database Systems, vol. 6, no. 2, p. 213–226, 1981.

[38] J. N. G. R. A. L. a. I. L. T. K. P. Eswaran, “The notions of consistency and predicate locks in a database system,” Communications of the ACM, vol. 19, no. 11, p. 624–633, 1976.

[39] P. A. A. a. J. D. Day, “A principle for resilient sharing of distributed resources,” in 2nd Int. Conf. on Software Engineering, 1976.

[40] M. Stonebraker, “Concurrency control and consistency of multiple copies of data in distributed INGRES,” IEEE Transactions on Software Engineering, vol. 5, no. 3, pp. 188-194, May 1979.

[41] L. a. R. O. C. Mohan, “Transaction management in the r* distributed database management system,” ACM Transactions on Database Systems, vol. 11, no. 4, p. 378–396, 1986.

[42] Tandem, “Nonstop SQL – a distributed high-performance, high-availability implementation of SQL,,” in Int. Workshop on High-Performance Transaction Systems,, 1987.

[43] Tandem, “A benchmark of NonStop SQL on the debit credit transaction,” in ACM SIGMOD Int. Conf. on Management of Data, 1988.

[44] Borr, “High performance SQL through low-level system integration,” in ACM SIGMOD Int. Conf. on Management of Data, 1988.

[45] H. a. J. P. Verjus, “An algorithm for maintaining the consistency of multiple copies,” in 1st Int. Conf. on Distributed Computing Systems, 1979.

[46] P. Reed, “Naming and synchronization in a decentralized computer system,” Cambridge, Mass., Sept., 1978.

[47] P. Reed, “Implementing Atomic Actions on Decentralized Data,” ACM Transactions on Computer Systems,, vol. 1, pp. 3-23, 1983.

[48] B. a. B. Mahbod, “Generalized version control in an object-oriented database,” in IEEE 4th Int. Conf. Data Engineering, February, 1988.

[49] H. T. C. a. W. Kim, “A unifying framework for versions in a CAD environment,” in Int. Conf. Very Large Data Bases, Kyoto, Japan, 1986.

[50] H. T. C. a. W. Kim, “Versions and change notification in an object-oriented database system,” in Design Automation Conference, 1988.

[51] K. a. K. Y. L. A. Chiu, “Comparing two-phase locking and optimistic concurrency control protocols in multiprocessor real-time databases,” in 5th International Workshop on Parallel and Distributed Real-Time Systems and 3rd Workshopon Object-Oriented Real-Time Systems, 1997.

[52] M. C. a. M. L. R. Agrawal, “Concurrency Control Performance Modeling: Alternatives and Implications,” ACM Transactions on Database Systems, Dec. 1987.

[53] K. R. a. S. C. J. A. Stankovic, “Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems,” IEEE Transactions on Computers, vol. 34, no. 12, pp. 1130-1143, Dec. 1985.

[54] J. L. P. a. A. Silberschatz, Operating System Concepts, Addison-Wesley Publishing Company, 1985.

[55] J. A. S. a. D. T. J. Huang, “On using priority inheritance in real-time databases,” in Twelfth. IEEE Real-Time Systems Symposium, 1991.

[56] S. D. a. L. Sha, “Sources of Unbounded Priority Inversion in Real-Time Systems and a Comparative Study of Possible Solutions,” ACM Operating Systems Review, p. 110–120, 1992.

[57] W. u. Haque, “Transaction Processing in Real-Time Database Systems,” 1993.

[58] N. Lynch, “Concurrency control for resilient nested transactions,” Advances in Computing Research, vol. 3, p. 335–376, 1986.

[59] N. L. a. M. Merrit, “Introduction to the theory of nested transactions,” Cambridge, Mass, 1986.

[60] N. L. M. M. a. W. E. W. A. Fekete, “Nested transactions and read/write locking,” in 6th ACM Symposium on Principles of Database Systems, San Diego, CA, 1987.

[61] J. K. L. a. A. Fekete, “Multi-granularity locking for nested transaction systems,” in MFDBS'91, 1991.

[62] N. L. M. M. a. W. E. W. A. Fekete, “Commutativity-based locking for nested transactions,” Journal of System Sciences, vol. 41, p. 65–156, 1990.

[63] S. N. M. a. B. C. S. K. Madria, “Formalization and correctness of a concurrency control algorithm for an open and safe nested transaction model using I/O automaton model,” in 8th International Conference on Management of Data (COMAD'97), Madras, India, 1997.

[64] R. a. T. Harder, “Concurrency control in nested transactions with enhanced lock modes for KBMSs,” in 6th International Conference on Database and Expert Systems Applications (DEXA'95), London, UK,, 1995.

[65] K. G. a. N. Lynch, “Nested transactions and quorum consensus,” ACM Transactions on Database Systems, vol. 19, p. 537–585, 1994.
[66] F. a. T. Kameda, “Concurrency control of nested transactions accessing B-trees,” in 8th ACM Symposium on Principles of Database Systems, 1989.
[67] S. N. M. a. B. C. S. K. Madria, “Formalization of linear hash structures using nested transactions and I/O automaton model,” in IADT 98, Berlin, Germany, 1997.
[68] S. L. A. a. A. M. A. M. Abdouli, “A System Supporting Nested Transactions in DRTDBSs,” in 1st International High-Performance Computing, September 2005.
[69] S. A. Majed Abdouli, “Scheduling distributed real-time nested transactions,” in IEEE ISORC, IEEE Computer Society, 2005.
[70] J. H. a. K. Ramamritham, “The prompt real-time commit protocol,” IEEE Transactions on Parallel and Distributed Systems, 2000.
[71] P. G. Ceri S, Distributed Database Principles and Systems, New York: McGraw-Hill, 1984.
[72] M. J. H. L. Chandy M, “Distributed deadlock detection,” ACM Trans Comput Syst. , vol. 1, no. 2, pp. 144-56, 1983.
[73] RC, “Some Deadlock Properties in Computer Systems,” ACM Comput Surveys, vol. 4, no. 3, pp. 179-96, 1972.
[74] R. A. Gray J, Transaction Processing: Concepts and Techniques, San Mateo: : Morgan Kaufmann Publ, 1993.
[75] R. M, “Hierarchical Deadlock Detection for Nested Transactions,” Distrib Comput. , vol. 4, no. 3, pp. 123-129, 1991.
[76] N. M. Sinha MK, “A Priority Based Distributed Deadlock Detection Algorithm,” IEEE Trans Softw Eng. , vol. 11, no. 1, pp. 67-80, 1985.
[77] M. Rukoz, A distributed solution for detecting deadlock in distributed nested transaction systems, vol. 392, J. R. M. Bermond, Ed., Berlin, Heidelberg: In Distributed Algorithms,Lecture Notes in Computer Science,Springer, 1989.
[78] S. C. M. Dong C. Shin, “A deadlock detection algorithm for nested transaction model,” Microprocessing and Microprogramming, vol. 28, no. 1, pp. 9-14, 1990.
[79] T. H. A. G. a. J. L. R. F. Resende, “Detection arcs for deadlock management in nested transactions and their performance,” in Advances in Databases, BNCOD ,Lecture Notes in Computer Science,Springer, Berlin,Heidelberg, 1997.
[80] S. K. Madria, “A Study of the Concurrency Control and Recovery Algorithms in Nested Transaction Environment.,” The Computer Journal, vol. 40, no. 10, pp. 630-639, 1997.
[81] N. L. a. W. E. W. Alan Fekete, “A serialization graph construction for nested transactions.,” in Ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS '90), New York, NY, USA, 1990.
[82] H. C. B. P. a. M. P. Weikum, “Multi-level recovery,” in 9th ACM Symposium on Principles of Database Systems, Nashville, TN, 1990.
[83] K. a. M. C. Rothermel, “ARIES/NT: a recovery method based on write-ahead logging for nested transactions,” in In Proceedings of the 15th international conference on Very large data bases (VLDB '89), San Francisco,CA,USA, 1989.
[84] H. D. L. B. P. H. a. S. P. Mohan, “ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging,” ACM Trans. Database Systems, vol. 17, no. 1, pp. 94-162, 1992.
[85] K. R. Theo Haerder, “Concepts for transaction recovery in nested transactions,” in ACM SIGMOD, 1987.
[86] Liskov, “Distributed programming in Argus,” Commun. ACM, vol. 31, no. 3, p. 300–312, 1988.
[87] &. D. M. &. H. M. &. J. P. &. S. R. &. W. W. Liskov, Argus Reference Manual, 1987.
[88] R. F. P. a. G. B. Z. Spector, “Camelot: a flexible, distributed transaction processing system,” in Digest of Papers, COMPCON Spring 88 Thirty-Third IEEE Computer Society International Conference, San Francisco, CA, USA,, 1988.
[89] R. J. L. a. W. F. A. P. Dasgupta, “The Clouds distributed operating system: functional description, implementation details and related work,” in The 8th International Conference on Distributed, San Jose, CA, USA, 1988.
[90] M. J. J. D. N. J.-L. B. a. C. P. W. H. Jessop, “The Eden transaction based file system,” in 2nd Symp. Reliability in Distributed Software and Database Systems, 1982.
[91] D. M. a. G. J. P. Erik T. Mueller, “A nested transaction mechanism for LOCUS,” in ninth ACM symposium on Operating systems principles (SOSP '83), New York, NY, USA, 1983.
[92] “Security Service Specification, Version 1.0,” 1996.

Mrs. Meenu is an Associate Professor in the department of Computer Science & Engineering at the Madan Mohan Malaviya University of Technology, Gorakhpur where she has been a faculty member since 2003. She is Chairperson of Women Cell as well as Women Welfare and

AntiHarassment Cell. She completed her M.Tech. at Madan Mohan Malaviya University of Technology. She has served as the Session Chair for UPCON-2018 (5th IEEE Uttar Pradesh Section International Conference). She is the author of 64 research papers, which have been published in various National & International Journals/Conferences. She is a reviewer of many International Journals/ Conferences and Editorial Board member of International Journals. She is also member of many Professional Societies. Her research interest lies in the area of Distributed Real Time Database Systems. She has collaborated actively with researchers in several other disciplines of computer science, particularly machine learning.