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

Hierarchical Deadlock Detection in Nested Transactions: A Depth-First Search Optimization Framework

Meenu

Submitted:13/03/2024 **Revised**: 28/04/2024 **Accepted**: 05/05/2024

Abstract Deadlock detection in nested transactions is crucial for maintaining system stability and ensuring efficient transaction processing. In nested transaction systems, deadlocks can occur not only between top-level transactions but also among subtransactions, resulting in intricate cycles that complicate resolution. Traditional methods, which often involve examining the entire Wait-For Graph (WFG) for cycles, tend to be resource-intensive and time-consuming. As the complexity of transactions increases, there is a growing need for efficient deadlock detection techniques to optimize resource utilization and enhance overall system performance. This research focuses on developing an efficient Depth-First Search (DFS)-based algorithm to address hierarchical deadlocks that arise from subtransaction dependencies. By modelling transaction relationships as a directed graph, where nodes represent transactions and edges denote dependencies, the proposed approach iteratively applies the DFS algorithm to identify cycles indicative of potential deadlocks. The algorithm not only detects deadlocks but also provides detailed insights into the involved transactions, the deadlock cycle, and indirect dependencies. Key performance metrics, such as execution time, memory usage, system throughput, response time, and success ratio, were computed during the detection process. Implemented and validated in a MATLAB simulation environment, the algorithm demonstrated significant improvements in deadlock detection efficiency within nested transactional systems. Results showed its effectiveness in identifying hierarchical deadlocks, with favourable performance in execution time, memory usage, and overall throughput, even in scenarios of varying transaction complexity. This research highlights the importance of efficient deadlock detection techniques in optimizing resource utilization and advancing concurrent system performance. The proposed DFS-based algorithm offers a robust solution for managing hierarchical deadlocks and lays the groundwork for future development of resilient and dependable transaction management systems.

INDEX TERMS: Cycle Detection, Deadlock Detection, Deadlock Resolution, Depth-First Search (DFS), Directed Graph Modelling, Hierarchical Deadlock Scenarios, Nested Transactions, Transaction Relationships.

I.INTRODUCTION

This section introduces the complexity of deadlock detection in nested transactions and proposes a Depth-First Search (DFS)-based approach using a graph model. It outlines the paper's key contributions, including the DFS algorithm and performance analysis, and summarizes the paper's structure, covering existing strategies, challenges, and experimental results.

In modern transactional systems, managing concurrency and ensuring data consistency are paramount. However, these systems often face challenges such as deadlocks, especially in environments with nested transactions. In a database (DB) system, transactions require locks on objects to prevent consistency anomalies due to concurrent access, which can lead to deadlocks where a cyclical sequence of transactions waits indefinitely for each other (T1 → T2 → ... → T1) (1) (2). Detecting and resolving these deadlocks is crucial, and one common method is the timeout approach, where the system aborts a transaction if it waits too long for a lock, assuming a deadlock (3). While simple to implement, this method is imprecise and often results in unnecessary aborts and restarts. A more precise strategy is the waits-for graph (WFG) method, where the system maintains a directed graph of waiting transactions, with nodes representing transactions and edges representing waiting situations (4). Deadlocks are detected by identifying cycles in the WFG, and the system resolves the deadlock by aborting one transaction in the cycle, removing its effects from the DB, and restarting it. The WFG method is precise for all transaction types and durations, making it a robust solution for deadlock detection. Because deadlocks are rare, the WFG cycle search is only initiated when likely, improving deadlock management performance (5) . Deadlock detection in nested transactions is more complex and costly than in flat transactions due to the need to account for transaction nesting, where deadlocks can occur among transactions in different hierarchies or among subtransactions within a single hierarchy. Unlike single-level transactions that rely on direct waits-for-lock relations, detecting deadlocks in nested transactions also requires maintaining waits-for-commit relations. This paper addresses the issue of deadlock detection and management in nested transactions, focusing on a Depth-First Search (DFS) approach. Nested transactions introduce a hierarchical structure where transactions can contain subtransactions, forming a complex graph of

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

dependencies. Traditional deadlock detection methods may not suffice in such scenarios due to the nested nature of transactions. Our approach involves modelling transaction relationships as a directed graph, where nodes represent transactions and edges represent dependencies between transactions and their subtransactions. We leverage DFS, a well-known graph traversal algorithm, to explore these transaction graphs and identify cycles indicative of potential deadlocks.

The key contributions of this paper include:

A. GRAPH-BASED MODELLING: We propose a graph-based model to represent nested transactions, allowing for a comprehensive analysis of transaction dependencies.

B. DFS-BASED DEADLOCK DETECTION: Our algorithm employs Depth-First Search to efficiently detect deadlock cycles within nested transactions.

C. METRICS AND ANALYSIS: We measure and analyse critical system metrics during deadlock detection, providing insights into system performance under deadlock scenarios.

To validate our approach, we implemented the algorithm in a MATLAB simulation environment and conducted extensive experiments using various transaction scenarios. The results demonstrate the effectiveness of our method in accurately detecting deadlocks, identifying involved transactions, and providing insights into potential resolution strategies.

The paper provides a comprehensive exploration of deadlock detection in nested transactions. Section 2 reviews existing deadlock detection strategies, establishing a foundation for understanding the field. Section 3 discusses the unique challenges of detecting deadlocks in nested transactions. Section 4 presents a DFS-based deadlock detection method designed for nested transactions, explaining how the algorithm handles hierarchical structures and relationships in a directed graph. Section 5 details the algorithm's steps, including graph representation, initialization, detection loops, and metric calculations. Section 6 analyses real-world results from MATLAB simulations, evaluating performance metrics like efficiency and scalability. Section 7 outlines future research directions and challenges, while Section 8 concludes with key findings and suggestions for further exploration in deadlock management for nested transactional environments.

In conclusion, this paper introduces a DFS-based approach for detecting deadlocks in nested transactions using a graph-based model, offering an efficient method to identify deadlock cycles. The approach is validated through MATLAB simulations, demonstrating its accuracy and scalability. The paper also outlines its structure, covering a review of existing strategies, challenges in nested transactions, the proposed method, performance analysis, and future research directions, providing a comprehensive framework for optimizing deadlock detection techniques in complex transactional systems.

II.RELATED WORK

This section explores deadlock detection strategies in nested transactions, highlighting the complexities introduced by hierarchical dependencies. It begins by outlining the adopted model of nested transactions, building upon Moss's framework and enhancing it to account for broader lock acquisition. The section then delves into deadlock detection mechanisms, addressing direct-wait and ancestor-descendant deadlocks within nested transaction systems. Finally, a comparative analysis of various deadlock detection strategies, including those proposed by Moss, Rukoz, Shin, and Rezende, is presented, evaluating their strengths and limitations based on different NT system requirements.
Deadlock detection in nested transactions has been a significant area of research due to the complexities introduced by hierarchical dependencies and concurrency control mechanisms. In nested transaction (NT) systems, deadlocks can arise not only between top-level transactions but also within the hierarchical structure of subtransactions.

A. A MODEL OF NESTED TRANSACTIONS

This section describes the nested transaction model, where transactions can have subtransactions, allowing all levels to acquire locks. It highlights the potential for inter-transaction and intra-transaction deadlocks, while supporting parallel execution and enhancing modularity for complex applications.

To understand the complexities of deadlock detection in nested transactions, it is essential to first establish the model of nested transactions that our paper employs. Our paper adopts the terminology outlined by Moss, where transactions can comprise multiple subtransactions, forming a potentially deep nested hierarchy (6). The top-level transaction (TL-transaction) is the root transaction not enclosed by any other, while transactions with subtransactions are termed parents, and their subtransactions are children. We also use the terms ancestors and descendants, where the ancestor (descendant) relation is the transitive closure of the parent (child) relation. The non-reflexive versions are referred to as superior (inferior). A transaction's descendants and their parent/child relationships constitute its hierarchy. Unlike Moss's model, where only leaf transactions can lock objects, ours permits every transaction to acquire locks, potentially leading to deadlocks across TL-transactions and within single transaction hierarchies. Our model

supports parent/child and sibling parallelism, allowing a parent transaction concurrent execution with its inferiors (non-strict), but it cannot commit until all inferiors have committed or aborted. The Nested Model, advantageous for complex applications, introduces complexity but is pivotal in engineering scenarios, offering operational abstractions and flexible ACID property handling. Moss's Nested Transaction Model, grounded in Bjork and Davies' "spheres of control," (7) (8) has two types: closed nested transactions, which confine subtransactions' impact to their parent's scope (9) (10) , and open nested transactions, which allow autonomous subtransaction execution and early leaf-level lock release (11) (12) (13) . Unlike flat transactions, nested transactions support independent subtransaction failure and rollback, enhancing performance through intra-transaction and intertransaction parallelism, and improving modularity, encapsulation, and security. These features make nested transactions ideal for real-time, intricate, and distributed applications (14). In nested transactions using the basic two-phase locking mechanism, two types of deadlocks can occur: inter-transaction deadlocks, which happen between two nested transactions similar to flat transactions, and intra-transaction deadlocks, which occur between subtransactions within the same nested transaction.

In conclusion, the nested transaction model enhances complex application management by allowing multiple subtransactions and enabling all levels to acquire locks. This structure supports parallel execution and improves modularity while addressing inter-transaction and intra-transaction deadlocks. Although it introduces complexity, its flexible ACID handling and independent rollback mechanisms make it ideal for real-time and distributed systems, advancing transaction processing efficiency and reliability.

B. DEADLOCK DETECTION IN NESTED TRANSACTIONS

This section addresses deadlock detection in nested transactions, emphasizing the adaptation of single-level transaction concepts to manage hierarchical complexities. It identifies two types of deadlocks—direct-wait and ancestor-descendant—and utilizes the Wait-For Graph (WFG) to detect cycles, ensuring effective deadlock management.

Having established the model of nested transactions, we now turn our attention to the mechanisms for detecting deadlocks within these systems. Effective deadlock detection in nested transactions requires extending concepts from single-level transactions and incorporating additional mechanisms to address the complexities introduced by hierarchical structures (6) (15) Deadlocks in nested transactions can be effectively managed by extending single-level transaction concepts and incorporating additional mechanisms, primarily through the representation of various waiting relations in the Wait-For Graph (WFG). Two main types of deadlocks are addressed:

1) DIRECT-WAIT DEADLOCKS

These occur when a transaction is blocked waiting for a lock held by another transaction, detected through direct-waits-for-lock relations in the WFG. A deadlock is identified if a cycle is present in these relations, such as mutual waiting between transactions A and B for lock release.

2) ANCESTOR-DESCENDANT DEADLOCKS

These occur 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 that all superiors of the waiting transaction also wait to commit until the deadlock is resolved. Combining direct-waits-for-lock and waits-for-commit relations proves effective in detecting deadlocks in nested transactions, offering a comprehensive approach to deadlock management.

In conclusion, effective deadlock detection in nested transactions is crucial for maintaining system integrity and performance. By extending the concepts from single-level transactions and utilizing the Wait-For Graph (WFG), this section highlights the identification of two primary deadlock types: direct-wait and ancestor-descendant. The proposed mechanisms ensure comprehensive detection and management of deadlocks, ultimately enhancing the reliability of nested transaction systems.

C. DEADLOCK DETECTION STRATEGIES IN NESTED TRANSACTIONS

This section analyses deadlock detection strategies in nested transactions, focusing on the challenges posed by hierarchical dependencies. It compares several approaches, highlighting their unique strengths and limitations. The choice of strategy depends on the specific needs of the system, balancing detection accuracy with resource efficiency.

Deadlock detection in nested transactions (NTs) presents unique challenges due to the hierarchical nature of transaction dependencies. In Nested Transactions (NT) systems using locks for concurrency control, committed transactions inherit their locks to their parent transactions instead of releasing them, requiring deadlock detection algorithms to account for these nested relationships. Various algorithms have been proposed to address these challenges, each offering distinct advantages and drawbacks. Below is a comparative analysis of prominent deadlock detection strategies in NT systems, highlighting

their key features, advantages, and disadvantages. The Table I below summarizes the comparison of Moss's, Rukoz's, Shin's, and Rezende's deadlock detection strategies:

TABLE I

OVERVIEW OF DEADLOCK DETECTION STRATEGIES FOR NESTED TRANSACTIONS

Author	Deadlock Detection Strategy	Advantages	Disadvantages
Moss	Follows edges in the wait-for graph to find cycles, addressing nested relationships in lock-based NT systems. (6) (3) (16)	Effective handling of nested relationships.	Potential performance overhead due to significant graph traversal.
Rukoz	Utilizes a distributed representative graph and hierarchical approach for deadlock detection in NT systems. (15) (17)	Distributed approach aligns with NT structure. Root process failures result in subtree aborts, reducing need for further detection steps.	Requires effective communication and coordination between distributed components. Dependency on root process functionality and reliability.
Shin	Proposes an edge-chasing algorithm for highly parallel NTs, addressing indirect waiting relationships. (18)	Avoids traversing transaction trees, constant message passing overhead regardless of nesting depth.	Prone to phantom deadlocks due to communication delays in distributed systems.
Rezende	Introduces detection arcs for deadlock management in NTs, reducing traversal of the waits-for graph (WFG). (19)	Enhances performance by traversing a minimal subset of WFG edges.	Requires careful handling of detection arcs to ensure accurate deadlock detection without overlooking cases.

Each deadlock detection strategy in nested transactions offers unique strengths and limitations. Moss's approach effectively handles nested relationships but may encounter performance overhead. Rukoz's distributed method aligns well with NT structures but requires robust communication and coordination. Shin's edge-chasing algorithm avoids tree traversal but needs to address phantom deadlock risks. Rezende's use of detection arcs enhances performance but demands careful management for accurate detection. The choice of deadlock detection strategy depends on the specific requirements and constraints of the NT system, balancing between effective detection and resource utilization. Table I summarizes the comparison of Moss's, Rukoz's, Shin's, and Rezende's deadlock detection strategies.

In conclusion, the examination of deadlock detection strategies in nested transactions highlights their unique strengths and limitations due to hierarchical dependencies. Each approach—Moss's, Rukoz's, Shin's, and Rezende's—offers different advantages and challenges, emphasizing the need for careful selection based on specific system requirements. Ultimately, the choice of strategy should balance detection accuracy with resource efficiency.

III.ISSUES AND CHALLENGES

This section outlines the challenges involved in detecting deadlocks within nested transactions. It underscores the necessity for sophisticated strategies to effectively manage dependencies, dynamic behaviours, and resource contention, ensuring reliable performance in transactional systems.

Detecting deadlocks in nested transactions involves numerous challenges due to the complex nature of dependencies, dynamic transaction behaviour, and resource contention. Below is a detailed analysis of the key issues in deadlock detection within nested transactions, along with potential resolutions for each issue. These issues and their resolutions are summarized in Table II:

TABLE II

DEADLOCK DETECTION IN NESTED TRANSACTIONS ISSUES AND REMEDIES

S.No.	Issues in Deadlock Detection in Nested Transactions	Description	Issues Resolution
1	Hierarchical Dependencies	Nested transactions introduce a hierarchical structure where	Develop advanced modelling techniques to accurately capture and

		transactions can contain subtransactions. This hierarchical nature leads to complex dependencies among transactions, making it challenging to accurately model and analyse deadlock scenarios.	analyse dependencies in hierarchical structures.
2	Cyclic Dependencies	Deadlocks arise due to cyclic dependencies where transactions wait for resources held by each other. In nested transactions, these cyclic dependencies can span across multiple levels of nesting, increasing the complexity of deadlock detection.	Implement algorithms that can detect and break cycles spanning multiple levels of nesting.
3	Dynamic Transaction Behaviour	Transactions in concurrent systems exhibit dynamic behaviour, with changes in transaction states and resource allocations over time. Managing dynamic behaviour while detecting and resolving deadlocks adds another layer of complexity.	Use adaptive algorithms that can respond to changes in transaction states and resource allocations in real-time.
4	Resource Contention	Resource contention is a fundamental issue in deadlock scenarios. Transactions compete for shared resources, and when a cyclic wait occurs, it can result in resource starvation and system deadlock.	Optimize resource allocation strategies to minimize contention and prevent cyclic waits.
5	Performance Impact	Deadlock detection and resolution mechanisms incur computational overhead, impacting system performance. Balancing accurate deadlock detection with minimal performance impact is a key challenge, especially in high-throughput transactional systems.	Design efficient detection mechanisms that balance accuracy and performance impact.
6	Transaction Rollback and Recovery	Resolving deadlocks often involves transaction rollback and	Develop robust rollback and recovery protocols that ensure data

		recovery strategies. In nested transactions, coordinating rollback and recovery across multiple levels of nesting while maintaining data consistency poses significant challenges.	consistency across all levels of nesting.
7	Scalability and Complexity	As the number of transactions and levels of nesting increases, the scalability and complexity of deadlock detection algorithms also escalate. Ensuring efficient and scalable deadlock detection mechanisms is essential for large-scale transactional systems.	Create scalable algorithms capable of handling large numbers of transactions and deep nesting levels.

Addressing these issues and challenges requires sophisticated deadlock detection algorithms tailored for nested transaction environments. Effective strategies for resource management, transaction coordination, and deadlock resolution are essential to build robust and reliable concurrent systems. The resolutions mentioned in Table II highlight the need for advanced modelling techniques, adaptive algorithms, efficient resource allocation strategies, robust rollback protocols, and scalable detection mechanisms to effectively manage and mitigate deadlocks in nested transactions. Implementing real-time detection and resolution systems will ensure that transactional systems maintain high performance and reliability, even under high-throughput conditions.

In conclusion, effective deadlock detection in nested transactions is essential for system reliability and performance. Addressing the complexities of hierarchical structures, dynamic behaviours, and resource contention requires advanced algorithms and strategies. By optimizing resource allocation and ensuring robust rollback protocols, we can improve detection and resolution efficiency, leading to resilient concurrent systems that maintain high performance in high-throughput environments.

IV.DFS BASED DEADLOCK DETECTION MODEL

This section introduces a DFS-based model for detecting deadlocks in concurrent systems and nested transactions. It organizes transactions into a directed graph and uses the Depth-First Search (DFS) algorithm to identify cycles indicating deadlocks. The model captures essential information about involved transactions and dependencies, supports performance metric analysis, and includes visualizations to aid in understanding and optimizing system stability.

Effective management of deadlocks is crucial in concurrent systems and nested transactions to uphold system stability and performance. In response, we've crafted a structured model that harnesses the Depth-First Search (DFS) algorithm specifically for detecting and addressing deadlocks within nested transactions. This DFS-based deadlock detection model stands as a pivotal framework designed for this precise purpose. It operates by structuring transactional relationships into a directed graph, where nodes represent transactions or subtransactions, and edges signify resource dependencies. The model's core lies in the DFS algorithm, systematically traversing this graph to identify cycles that indicate potential deadlocks. As DFS explores paths in the graph, revisiting nodes already in the current path signals the presence of a cycle, highlighting a deadlock scenario. This methodical approach ensures precise deadlock detection, enabling the model to gather critical information like involved transactions, the deadlock cycle, and contributing dependencies. Additionally, the model facilitates metric analysis, providing insights into deadlock occurrences, system behaviour under varying conditions, and performance optimizations. Visualizations of the transactional graph and deadlock scenarios further aid in understanding and optimizing system stability in complex concurrent environments.

In conclusion, the DFS-based deadlock detection model effectively identifies deadlocks in concurrent systems and nested transactions by leveraging the Depth-First Search algorithm to detect cycles in directed graphs. This

approach enhances detection accuracy and offers valuable insights for optimizing system stability and performance.

V.PROPOSED ALGORITHM

This section outlines a proposed DFS-based algorithm for detecting deadlocks in nested transactions, focusing on enhancing system stability and performance. It represents transactions as a directed graph, initializes metrics, and detects deadlocks through DFS by identifying cycles. The algorithm calculates average performance metrics and visualizes results, ensuring precise deadlock detection and comprehensive analysis in concurrent systems.

In the realm of nested transactions, managing deadlocks is crucial for ensuring system stability and performance. We propose an implementation of the Depth-First Search (DFS) algorithm tailored for detecting deadlocks within nested transactions. This algorithm efficiently traverses the transaction graph, identifying cycles that signal potential deadlocks.

A. ALGORITHM

This algorithm is structured to systematically identify and handle deadlocks in nested transactions using Depth-First Search (DFS). Below is a detailed breakdown of each step involved in the process:

STEP 1: GRAPH REPRESENTATION

1) Create a directed graph where nodes represent transactions, and edges represent resource dependencies.

STEP 2: INITIALIZATION

1) Initialize arrays to store metrics such as execution times, memory usage, system throughput, response times, and success ratios for each iteration.

2) Set the number of transactions and iterations.

STEP 3: GRAPH SETUP

1) Initialize a directed graph (digraph) to represent transaction relationships.

2) Add edges to the graph to simulate transaction dependencies and subtransactions.

STEP 4: ITERATION LOOP

1) For each iteration:

a) Add subtransactions to the graph as defined.

b) Call the detectDeadlock function to detect deadlocks and calculate metrics.

c) Store metrics for the current iteration.

STEP 5: DEADLOCK DETECTION FUNCTION (DETECTDEADLOCK)

1) Initialize containers for visited nodes and nodes in the current path (stack) using containers.Map.

2) Start a timer (tic) to measure execution time.

3) Perform DFS for each node in the graph to detect cycles:

a) Mark nodes as visited and add them to the stack.

b) Explore successors of the current node recursively.

c) If a cycle is found (i.e., a node is visited that's already in the stack), a deadlock is detected.

d) Display deadlock-related information like involved transactions, deadlock cycle, and indirect dependencies contributing to deadlock.

4) Stop the timer (toc) and calculate execution time.

5) Return metrics (dummy values if no deadlock is detected).

STEP 6: METRIC CALCULATION

1) Calculate overall averages for metrics across all iterations.

2) Display the overall summary of metrics.

STEP 7: GRAPH PLOTTING

1) Plot individual graphs for each metric (execution time, memory usage, system throughput, response time, success ratio).

2) Plot the transaction tree graph.

The methodical nature of this approach guarantees precise deadlock identification and thorough metric evaluation within nested transactions. This structured framework not only enhances transactional dependency management but also facilitates the optimization of system performance, making it indispensable for developers and researchers navigating the complexities of concurrent systems.

In conclusion, the proposed DFS-based algorithm provides a systematic and efficient approach for detecting deadlocks in nested transactions, essential for maintaining stability and performance in concurrent systems. By representing transactions as a directed graph and utilizing Depth-First Search to identify cycles, this algorithm ensures accurate deadlock detection while also facilitating detailed metric analysis and visualization. The structured framework enhances the management of transactional dependencies and aids in optimizing overall system performance, making it a valuable tool for developers and researchers in the field of concurrent transaction management.

VI.IMPLEMENTATION AND RESULTS

This section outlines the implementation and evaluation of a Depth-First Search (DFS)-based deadlock detection algorithm for nested transactional systems, analysing configurations of 5 to 30 parent transactions over ten

iterations. We validated our approach using MATLAB simulations, demonstrating its effectiveness in detecting and managing deadlocks in nested transactions. It discusses key performance metrics—execution time, memory usage, system throughput, success ratio, and response time—that are essential for assessing the effectiveness and efficiency of deadlock detection protocols. The section emphasizes the necessity of a comprehensive analysis to understand system performance under varying workloads and transaction complexities, ultimately highlighting the significance of optimizing resource management and performance monitoring in deadlock detection.

We conducted simulations to evaluate the performance of our DFS-based deadlock detection algorithm in nested transactional systems, with varying numbers of transactions and different levels of nesting. The experimental setup included a simulated environment with 5, 10, 15, 20, 25, and 30 parent transactions, each with two subtransactions, and nested transaction depths ranging from 1 to 5 levels. Metrics were recorded over 10 iterations for each configuration. The total number of transactions considered in each model is as follows: 5 parent transactions with 2 subtransactions each, resulting in 15 transactions (referred to as the "05 transaction model"); 10 parent transactions with 2 subtransactions each, resulting in 30 transactions (referred to as the "10 transaction model"); 15 parent transactions with 2 subtransactions each, resulting in 45 transactions (referred to as the "15 transaction model"); 20 parent transactions with 2 subtransactions each, resulting in 60 transactions (referred to as the "20 transaction model"); 25 parent transactions with 2 subtransactions each, resulting in 75 transactions (referred to as the "25 transaction model"); and 30 parent transactions with 2 subtransactions each, resulting in 90 transactions (referred to as the "30 transaction model"). This model structure allows for a comprehensive analysis of system performance across varying levels of complexity and workload, providing insights into how the system handles increasing numbers of transactions. When evaluating the effectiveness of deadlock detection in nested transactions using Depth-First Search (DFS), it's crucial to consider a range of performance metrics that provide insights into the protocol's capabilities. Key metrics such as average system throughput, success ratio, response time, execution time, and memory usage were analysed to assess the protocol's performance and efficiency. DFS traverses transactional relationships to identify cycles indicative of potential deadlocks. Evaluating system throughput helps gauge the protocol's ability to detect deadlocks within a specified timeframe, ensuring timely resolution and system stability. The success ratio metric reflects the protocol's effectiveness in resolving deadlocks and maintaining transaction reliability. A higher success ratio indicates that a larger proportion of transactions are successfully completed without encountering deadlocks, showcasing the protocol's robustness in handling concurrent transactions, and minimizing disruptions due to deadlock situations. Response time measures the system's responsiveness to detecting and handling deadlock situations. A lower response time indicates quicker detection and resolution of deadlocks, which is crucial for maintaining system performance and user satisfaction. Execution time reflects the efficiency of deadlock detection and resolution processes. A protocol with lower execution times for deadlock handling tasks can manage concurrent transactions more effectively, minimizing disruptions caused by deadlock situations. Assessing memory usage is crucial for scalability and efficient resource management during deadlock handling processes. A protocol that optimizes memory usage while effectively managing deadlock scenarios can enhance overall system performance and reliability. Experimental results and performance trends, summarized in Tables III to VIII, showcase the protocol's strengths and areas for improvement in handling deadlock scenarios within nested transactions. FIGURES 1-11 illustrate the performance trends of the nested transaction systems under various configurations, paving the way for further research and optimization in deadlock detection.

In conclusion, this section presents the framework for implementing a Depth-First Search (DFS)-based deadlock detection algorithm tailored for nested transactional systems. We established a simulated environment with varying configurations of parent transactions and subtransactions, providing a structured approach to assess the protocol's performance. This framework sets the foundation for further exploration and optimization in the field of deadlock detection within complex transactional systems.

A. Performance Metrics

This section outlines essential performance metrics for evaluating deadlock detection protocols in nested transactions. These metrics, summarized in Table III, include system throughput, success rates, response times, execution durations, and memory utilization. Each metric provides valuable insights into the system's efficiency and effectiveness, enabling stakeholders to identify areas for improvement and optimize transaction management. By measuring these indicators, a comprehensive assessment of the deadlock detection system's overall performance capacity can be achieved.

In detecting deadlocks in nested transactions, various performance metrics are employed to gauge the effectiveness and efficiency of the detection protocols. These metrics offer a comprehensive view of different performance aspects, enabling stakeholders to make

informed decisions and implement improvements. Table III summarizes the key performance metrics used in this evaluation. These metrics are crucial for assessing the efficiency, success rate, detection time, and overall performance capacity of the deadlock detection system.

TABLE III

PERFORMANCE METRICS FOR DEADLOCK DETECTION IN NESTED TRANSACTIONS

S.No.	Performance Metric	Description
1.	Average System Throughput	Measures the average number of transactions processed per unit of time, indicating system capacity.
2.	Average Success Ratio	Average ratio of successful transactions to total transactions submitted, showing system success rate.
3.	Average Response Time	Average elapsed time from command initiation to completion, assessing transaction performance.
4.	Average Execution Time	Average typical duration for completing each transaction, reflecting system efficiency.
5.	Average Memory Usage	Average amount of memory utilized per transaction, indicating system memory efficiency.

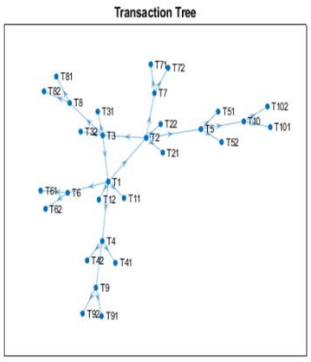

FIGURE 1. Five parent node transaction structure

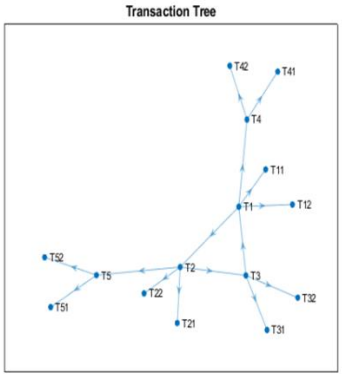

FIGURE 2. Ten parent node transaction structure

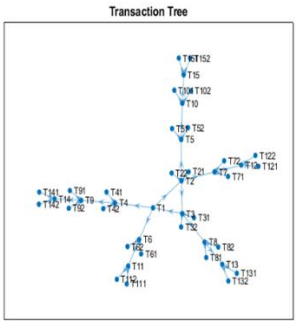

FIGURE 3. Fifteen parent node transaction structure

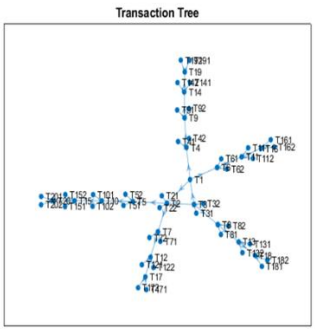

FIGURE 4. Twenty parent node transaction structure

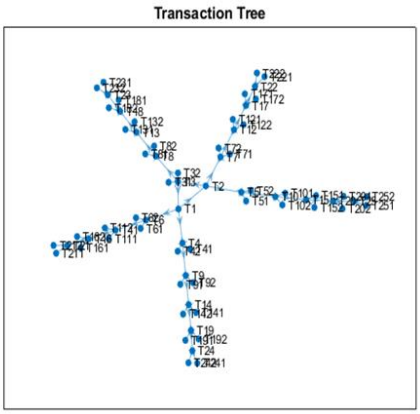

FIGURE 5. Twenty-five parent node transaction structure

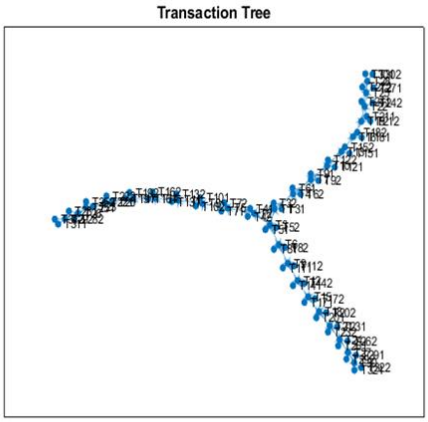

FIGURE 6. Thirty parent node transaction structure

In conclusion, the performance metrics discussed are vital for evaluating deadlock detection protocols in nested transactions. By analysing system throughput, success rates, response times, execution durations, and memory utilization, stakeholders can gain valuable insights into system efficiency and effectiveness. This assessment facilitates informed decision-making and identifies areas for improvement, ensuring optimized performance and reliability in transaction management as complexity increases.

B. Performance Metrics Analysis

This section examines the essential performance metrics used to evaluate deadlock detection protocols in nested transactions, focusing on execution time, memory usage, system throughput, success ratio, and response time. Analysing these metrics provides insights into the system's efficiency, reliability, and resource utilization, which are critical for informed decision-making and improvements in deadlock detection strategies. The subsequent analysis highlights trends and patterns in these metrics, emphasizing the importance of understanding their implications for overall system performance in managing deadlocks effectively. Before delving into the analysis of these performance metrics, it is essential to highlight their significance in evaluating deadlock detection protocols in nested transactions. These metrics, including Average System Throughput, Average Success Ratio, Average Response Time, Average Execution Time, and Average Memory Usage, provide a comprehensive view of the system's efficiency, success rate, detection performance, and resource utilization. Understanding these metrics is crucial for making informed decisions and implementing improvements in deadlock detection protocols for nested transactions. The following section will provide a detailed analysis of each metric, shedding light on their impact and implications for system performance.

1) ANALYSIS OF EXECUTION TIME TRENDS

This section analyses average execution time to evaluate the system's processing efficiency across various transaction workloads. It details the recorded execution times for different transaction volumes, illustrating how execution time scales with the number of transactions. The analysis provides insights into how execution time impacts overall transaction management and efficiency in the system.

Examining the average execution time provides essential insights into the system's processing efficiency across various transaction workloads. Table IV details the recorded average execution times for different transaction volumes over ten iterations, offering a comprehensive view of how execution time scales with the number of transactions. This information is visually represented in FIGURE 7, depicting the execution time trends, and highlighting any significant patterns or outliers within the transaction workload spectrum.

TABLE IV EXECUTION TIME FOR SIMULATION

Number of Parent Transactions	Average Execution Time (seconds)
5	0.0411
10	0.107

15	0.069
20	0.1159
25	0.075
30	0.0927

The analysis of average execution time versus the number of transactions reveals a non-linear pattern in system performance. Initially, there's a significant increase from 5 to 10 parent transactions, indicating overheads and resource contention. This is followed by a decrease from 10 to 15 parent transactions, possibly due to optimization or improved resource management. However, from 15 to 20 parent transactions, there's another increase, suggesting increased overhead. The pattern continues with fluctuations, highlighting the impact of resource contention and transaction complexity on performance.

In conclusion, the analysis of execution time trends highlights the intricate relationship between transaction workload and system processing efficiency. By examining the variations in average execution time across different transaction volumes, we gain valuable insights into the performance dynamics of the system. The observed fluctuations underscore the importance of effective resource management and optimization strategies to mitigate overhead and enhance overall efficiency in handling nested transactions. Continuous monitoring and adaptive management practices will be essential for maintaining optimal performance as transaction workloads evolve.

2) ANALYSIS OF MEMORY USAGE

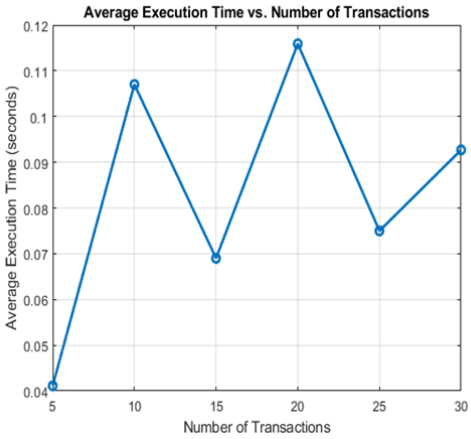

FIG 7. Execution time for simulation

This section analyses the system's memory usage under different transaction workloads, providing insights into its scalability and resource management. It highlights the system's ability to efficiently handle varying transaction volumes without requiring additional memory, indicating effective memory allocation and optimization mechanisms.

Analysing average memory usage offers valuable insights into the system's memory requirements under different transaction workloads. Table V shows the recorded average memory usage for varying numbers of transactions over ten iterations, illustrating how memory usage scales with transaction volume. This information is visually represented in FIGURE 8, providing a clear depiction of memory usage trends, and highlighting any significant patterns or fluctuations across different transaction volumes.

TABLE VMEMORY USAGE FOR SIMULATION

Number of Parent Transactions	Average Memory Usage (MB)
5	67.4522

10	67.4522
15	67.4522
20	67.4522
25	67.4522
30	67.4522

The analysis of average memory usage versus the number of transactions reveals a constant memory usage of 67.4522 MB across all transaction counts from 5 to 30 parent transactions. This suggests efficient memory management and allocation mechanisms in the system, indicating scalability and optimized memory usage. The system's ability to handle increasing transactions without requiring additional memory resources reflects positive scalability and efficiency in memory utilization, potentially through techniques like memory pooling or efficient memory reuse.

In conclusion, the system demonstrates effective memory management and scalability by maintaining a constant memory usage across increasing transaction volumes. This stability indicates the use of efficient optimization mechanisms, ensuring that the system can handle higher workloads without requiring additional memory resources, ultimately contributing to its overall performance and efficiency.

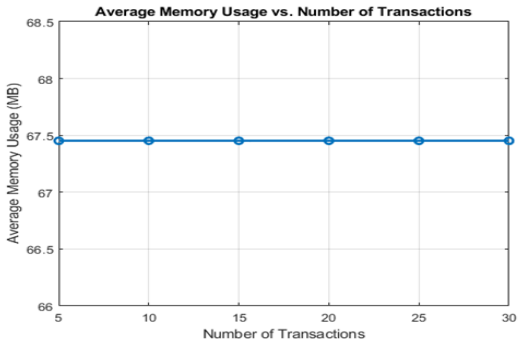

FIGURE 8. Memory Usage for simulation

3) ANALYSIS OF SYSTEM THROUGHPUT

This section evaluates system throughput across varying transaction workloads, providing insights into the system's processing capacity. It highlights the system's ability to maintain consistent throughput under different transaction volumes, suggesting effective load balancing, resource management, and scalability. The analysis emphasizes the system's capability to handle increasing workloads efficiently without performance degradation, indicating robust optimization mechanisms in place.

Evaluating the average system throughput offers crucial insights into the system's processing capacity across varying transaction workloads. Table VI compiles the average throughput values recorded for different transaction volumes over ten iterations, demonstrating how throughput varies with transaction volume. This data is visually presented in FIGURE 9, providing a graphical representation of throughput trends, and emphasizing any notable fluctuations or trends observed across the transaction volume range.

TABLE VI SYSTEM THROUGHPUT FOR SIMULATION

Number of Parent Transactions	Average System Throughput (TPS)
5	46.9724
10	46.9724
15	46.9724
20	46.9724
25	46.9724
30	46.9724

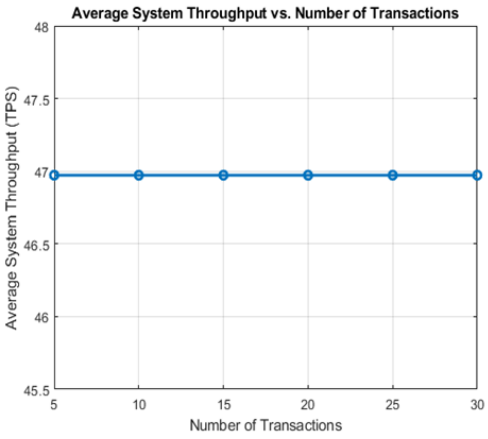

FIGURE 9. System Throughput for simulation

The analysis of average system throughput versus the number of transactions shows a consistent throughput of 46.9724 transactions per second (TPS) across all transaction counts from 5 to 30 parent transactions. This indicates a stable and well-optimized system capable of handling varying loads efficiently. The constant throughput suggests effective load balancing, resource management, and scalability, ensuring steady performance without degradation even as transaction volumes increase.

In conclusion, the consistent throughput across varying transaction volumes demonstrates the system's strong scalability and efficient resource management. The stability in processing capacity highlights the system's robustness in handling increased workloads without experiencing any performance degradation, underscoring the effectiveness of its optimization mechanisms and load balancing strategies.

4) ANALYSIS OF SUCCESS RATIO

This section focuses on assessing the average success ratio to evaluate the system's reliability and transaction completion rates across varying transaction workloads. It provides insights into the system's performance by detailing how success rates fluctuate with transaction volume. The analysis emphasizes the importance of maintaining a high success ratio in ensuring robust functionality and a positive user experience.

Assessing the average success ratio provides crucial insights into the system's reliability and transaction completion rates across varying transaction workloads. Table VII aggregates the average success ratio values recorded for different transaction volumes over ten iterations, offering a comprehensive view of success rates across transaction volumes. This data is visually depicted in FIGURE 10, graphically presenting success ratio trends, and highlighting any significant patterns or variations observed in the system's performance concerning transaction completions.

TABLE VII SUCCESS RATIO FOR SIMULATION

Number of Parent Transactions	Average Success Ratio
5	0.5748
10	0.5748
15	0.5748
20	0.5748
25	0.5748

30	0.5748

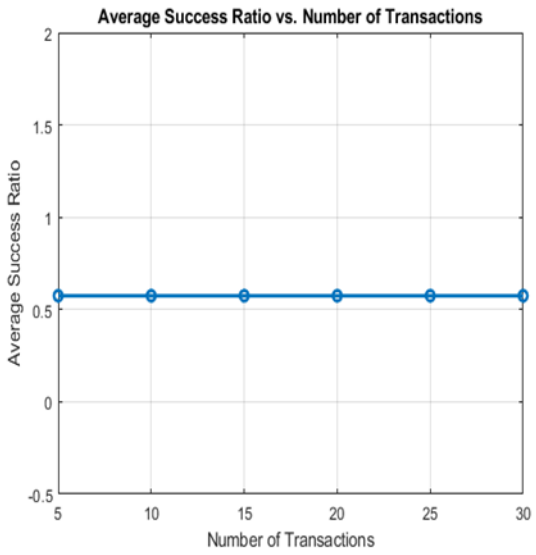

FIGURE 10. Success Ratio for simulation

The analysis of the average success ratio versus the number of transactions reveals a constant value of 0.5748 across all transaction counts from 5 to 30 parent transactions. This indicates a stable and reliable system with consistent success rates in transaction handling, regardless of workload variations. The system's predictable performance suggests robust functionality and user experience.

In conclusion, the consistent average success ratio across all transaction volumes highlights the system's reliability and effective transaction management. This stability in success rates, despite varying workloads, reflects the system's robust functionality and efficiency in handling transactions. Such predictable performance is crucial for ensuring a positive user experience, reinforcing the importance of maintaining high success rates in transaction processing systems.

5) ANALYSIS OF RESPONSE TIME

This section evaluates the average response time to assess the system's responsiveness under varying transaction workloads. It highlights the importance of consistent response times for user experience and system reliability, emphasizing effective resource allocation and stable performance in handling requests. The analysis underscores the necessity for ongoing monitoring and performance optimization to ensure scalability and sustained performance across different usage patterns.

Analysing the average response time provides insights into the system's responsiveness under varying transaction workloads. Table VIII summarizes the recorded average response time values for different transaction volumes over ten iterations, showing how response time changes with transaction volume. This data is visually represented in FIGURE 11, offering a graphical depiction of response time trends, and highlighting any significant changes or patterns observed in the system's responsiveness across various transaction volumes.

TABLE VIII RESPONSE TIME FOR SIMULATION

Number of Parent Transactions	Average Response Time (seconds)
5	2.7506
10	2.7506
15	2.7506
20	2.7506

25	2.7506
30	2.7506

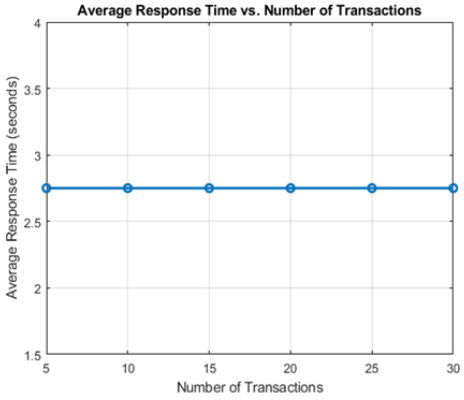

FIGURE 11. Response Time for simulation

The analysis of average response time versus the number of transactions shows a constant value of 2.7506 seconds across all transaction counts from 5 to 30 parent transactions. This indicates a stable and predictable system performance, crucial for user experience and system reliability. The constant response time suggests effective resource allocation and consistent performance in handling requests, highlighting system stability. However, ongoing monitoring, load testing, and performance optimization are necessary to ensure scalability and sustained performance under varying workloads and usage patterns.

In conclusion, the analysis of average response time reveals a stable performance across different transaction volumes, indicating effective resource management and consistent responsiveness. This predictability is vital for maintaining user satisfaction and system reliability. However, to support scalability and adapt to varying workloads, it is essential to implement ongoing monitoring and performance optimization strategies. Ensuring that the system can handle increased demands while maintaining responsiveness will be crucial for its long-term effectiveness and user experience.

In conclusion, the comprehensive analysis of performance metrics for deadlock detection protocols in nested transactions has provided critical insights into the system's efficiency, reliability, and resource management. By examining key metrics such as execution time, memory usage, system throughput, success ratio, and response time, we established a robust framework for evaluating overall system performance. The findings reveal a non-linear relationship between transaction workload and execution time, emphasizing the necessity of effective resource management and optimization strategies to mitigate overhead and enhance processing efficiency. Notably, the constant memory usage across varying transaction volumes showcases effective memory allocation mechanisms, reflecting the system's scalability and its capacity to handle increased workloads without additional memory requirements. Furthermore, the stable system throughput signifies the success of load balancing and resource management, ensuring consistent processing capacity even with rising transaction volumes. The high and consistent success ratio across different transaction counts reinforces the system's reliability and its capability to maintain a positive user experience. Additionally, the stable average response time underscores the system's predictable performance, which is crucial for user satisfaction and operational reliability.

Overall, these outcomes provide valuable insights for stakeholders seeking to optimize transaction management systems. By identifying key performance indicators and understanding their implications, stakeholders can make informed decisions to enhance the efficiency and effectiveness of deadlock detection protocols in nested transactions, ensuring scalability and reliability as complexity increases. Continuous monitoring and adaptive management practices will be essential for sustaining optimal performance amid evolving transaction workloads.

The detailed analysis further underscores the effectiveness of the Depth-First Search (DFS) algorithm in managing nested transactions, highlighting correlations between workload and processing efficiency metrics. The constant values in memory usage, system throughput, success ratio, and response time suggest a stable and predictable system. However, the observed non-linear trend in average execution time points to the system's sensitivity to workload changes, indicating potential overhead and resource contention periods.

Overall, the system exhibits reliability, scalability, and effective resource management in detecting and resolving deadlocks. However, continuous monitoring and optimization efforts are necessary to ensure consistent performance and to address fluctuations in execution time, thus enhancing the overall robustness of the deadlock detection framework.

VII.FUTURE DIRECTIONS AND CHALLENGES

This section addresses the challenges and opportunities in deadlock management within transactional systems, emphasizing the need for further development to enhance system resilience and adaptability. It highlights the necessity for ongoing research and collaboration to advance deadlock detection and resolution mechanisms.

The current state of deadlock management in transactional systems presents both challenges and opportunities for improvement. While existing deadlock detection algorithms and strategies have made significant advancements in mitigating deadlocks, there are several areas where further development and innovation are needed to address emerging complexities and enhance system resilience. In light of these considerations, the following future directions and challenges outline key areas for research and development in deadlock resolution strategies and transactional system management.

A. ADVANCED DEADLOCK RESOLUTION STRATEGIES

Explore transaction prioritization, dynamic resource allocation, and adaptive scheduling. Automate deadlock resolution to minimize manual intervention and enhance resilience.

B. DYNAMIC GRAPH ANALYSIS

Extend algorithms to handle real-time updates and changes in transaction graphs. Implement adaptive analysis techniques to maintain detection efficiency.

C. OPTIMIZATION AND PARALLELISM

Improve algorithm efficiency and scalability for large-scale graphs. Utilize parallel processing and distributed computing to accelerate detection.

D. MACHINE LEARNING INTEGRATION

Use machine learning to enhance prediction and prevention capabilities. Train models on historical data for early deadlock detection.

E. FAULT TOLERANCE AND RECOVERY

Enhance fault tolerance to recover from deadlock-induced failures. Develop automated recovery strategies to minimize downtime and ensure data consistency.

F. REAL-TIME MONITORING AND ANALYSIS

Implement real-time monitoring tools and integrate anomaly detection to identify potential deadlocks early.

G. CROSS-PLATFORM COMPATIBILITY

Ensure deadlock detection solutions are compatible across various platforms. Address transaction coordination and resource management challenges in distributed environments.

H. SECURITY AND PRIVACY CONSIDERATIONS

Implement encryption and secure protocols to protect deadlock-related data. Ensure data privacy and access control within detection mechanisms. Addressing these future directions requires ongoing research, collaboration, and innovation to develop robust, efficient, and adaptable deadlock detection and resolution mechanisms for modern transactional systems.

In conclusion, the future directions and challenges in deadlock management within transactional systems underscore the importance of continual advancement and innovation. By focusing on enhancing resolution strategies, improving adaptability to dynamic environments, and integrating cutting-edge technologies, the field can better address the complexities of modern systems. Ongoing research, collaboration, and the development of robust mechanisms will be essential for ensuring that deadlock detection and resolution processes remain effective and efficient. Ultimately, these efforts will contribute to the resilience and reliability of transactional systems, enabling them to meet the demands of increasingly complex and evolving operational landscapes.

VIII.CONCLUSION

This section concludes that the advanced deadlock management approach proposed in this paper, which employs a Depth-First Search (DFS)-based detection mechanism, significantly enhances the reliability and performance of nested transaction systems. By introducing relations such as direct-waits-for-lock, waits-for-commit, and indirect-waits-for-lock, the methodology facilitates early deadlock detection and prevents deadlock cycles, ensuring uninterrupted transaction progress. The approach minimizes the need for exhaustive cycle searches within the transaction graph, leading to substantial improvements in processing efficiency. Performance evaluations highlight stable execution times, consistent memory usage, reliable throughput, predictable response times, and steady success rates across various transaction volumes, demonstrating the effectiveness of the proposed method in resource management and system reliability. This research establishes a robust foundation for efficient deadlock management in nested transactions, with future efforts directed towards optimizing execution times and exploring additional strategies to further enhance the resilience, scalability, and adaptability of transaction management systems.

References

[1] P. A. Bernstein VHaNG. Concurrency Control and Recovery in Database Systems: Addison Wesley; 1987.
[2] Ceri S PG. Distributed Database Principles and Systems: New York: McGraw-Hill; 1984.
[3] Chandy M MJHL. Distributed deadlock detection. ACM Trans Comput Syst.. 1983; 1(2): 144-56.
[4] RC H. Some Deadlock Properties in Computer Systems. ACM Comput Surveys. 1972; 4(3): 179-96.
[5] Gray J RA. Transaction Processing: Concepts and Techniques San Mateo: : Morgan Kaufmann Publ; 1993.
[6] Moss JEB. Nested Transactions: An Approach to Reliable Distributed Computing. Cambridge, MA:; April 1981.
[7] Bjork LA. Recovery scenario for a DB/DC system. In ACM Annual Conference; 1973. p. 142–6.
[8] Davies CT. Recovery semantics for a DB/DC system. In ACM Annual Conference; 1973. p. 136–41.
[9] Salem HGMaK. Sagas. In ACM SIGMOD International Conference on Management of Data; 1987. p. 249–259.
[10] Karabatis G. Nested Transaction Models. In Liu L. ÖM, editor. Encyclopedia of Database Systems; 2017; New York: Springer.
[11] Medjahed MOAKE. Generalization of ACID Properties. In Liu L. ÖMT, editor. Encyclopedia of Database Systems; 2009; Boston, MA: Springer.
[12] Buchmann A. Open Nested Transaction Models. In Liu L. ÖM, editor. Encyclopedia of Database Systems; 2016; New York: Springer.
[13] A. EI-Sayed HSHaMEES. Effect of shaping characteristics on the performance of nested transactions. Information and Software Technology. 2001; 43(10): 579-590.
[14] Rothermel THaK. Concurrency Control Issue in Nested Transactions. VLDB J. 1993; 2(1): 39-74.
[15] M R. Hierarchical Deadlock Detection for Nested Transactions. Distrib Comput.. 1991; 4(3): 123-129.
[16] Sinha MK NM. A Priority Based Distributed Deadlock Detection Algorithm. IEEE Trans Softw Eng.. 1985; 11(1): 67-80.
[17] Rukoz M. A distributed solution for detecting deadlock in distributed nested transaction systems Bermond J,RM, editor. Berlin, Heidelberg: In Distributed Algorithms,Lecture Notes in Computer Science,Springer; 1989.
[18] Dong C. Shin SCM. A deadlock detection algorithm for nested transaction model. Microprocessing and Microprogramming. 1990; 28(1): 9-14.
[19] Rezende F&HT&GA&LJ. Detection arcs for deadlock management in nested transactions and their performance; 2006.

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.