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

Transaction Models: A Comprehensive Review of Evolution, Challenges, And Future Prospects

Meenu

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

Abstract: This review paper explores the evolution of transaction models in database systems, highlighting their foundational principles and advancements. It begins by examining the limitations of classical ACID-based models in handling modern, complex database systems, including distributed, real-time, and long-duration transactions. Advanced models, such as nested, distributed, and real-time transactions, are introduced to address scalability, fault tolerance, and performance challenges. Key issues such as deadlock management and scalability are explored, alongside emerging trends like machine learning for optimization and blockchain for secure processing. The paper identifies future research directions, including energy-efficient management, AI-driven optimizations, and support for heterogeneous environments, ultimately emphasizing the need for adaptive transaction frameworks to meet evolving demands in database systems.

Index Terms: *ACID Properties, Distributed Systems, Nested Transactions, Real-Time Transactions. Transaction Models.*

I.INTRODUCTION

This section outlines the evolution of data management systems, from manual filing systems to Centralized Database Systems (CDBS) and Distributed Database Systems (DDBS), addressing scalability and global access. It also introduces Real-Time Database Systems (RTDBS) and Distributed Real-Time Database Systems (DRTDBS) for time-sensitive applications. The section emphasizes the importance of transaction models, adhering to ACID properties, in ensuring data integrity and reliability in these advanced systems. Additionally, it provides an overview of the paper's structure, which includes discussions on classical and advanced transaction models, key challenges, and future research directions.

The evolution of data management systems has significantly transformed the way information is stored, accessed, and processed. Initially, manual filing systems were used to organize information, but they had several limitations. These early systems lacked integrated data definitions, were prone to redundancy, and were inefficient when it came to scalability and data control. As data management became more complex, these deficiencies highlighted the need for more robust solutions, leading to the development of Centralized Database Systems (CDBS). CDBS centralized data management, ensuring greater consistency, reducing redundancy, and offering better control over data [1]. These systems provided a more structured and systematic approach to data storage and retrieval, allowing organizations to manage vast amounts of data more effectively. However, as businesses and applications expanded beyond localized environments, the limitations of CDBS became apparent, particularly in terms of geographical reach and scalability. The growing need for accessing and managing data from multiple locations led to the emergence of Distributed Database Systems (DDBS). DDBS allow databases to be spread across multiple locations, providing users with global access to data. These systems address the challenge of data distribution by enabling databases to be physically distributed but logically integrated, ensuring that users across various geographical regions can access the same data seamlessly [2]. DDBS support improved data availability and fault tolerance, although they introduce challenges in synchronization, data consistency, and transaction management. At the same time, advancements in technology began to push the boundaries of data processing beyond simple storage and retrieval. The demand for real-time applications, which require guaranteed timing constraints and immediate processing of data, led to the rise of Real-Time Database Systems (RTDBS) [3]. RTDBS are designed to handle time-sensitive operations such as air traffic control, financial markets, and industrial automation, where delays in data processing or missed deadlines could have severe consequences [4] [5]. These systems integrate timing constraints into transaction management, ensuring that operations are executed within specified time frames [6]. The need for both real-time and distributed capabilities gave rise to Distributed Real-Time Database Systems (DRTDBS). DRTDBS combine the features of RTDBS with the benefits of distributed architectures, enabling time-sensitive transactions to be processed across multiple distributed locations [7]. These systems are particularly suited for applications that require global coordination while adhering to strict timing constraints, such as in

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

defence systems, telecommunications, and large-scale industrial operations.

To ensure the effective and reliable operation of these systems, transaction models have been developed to manage the execution of operations within databases [8]. These transaction models define how operations are grouped, executed, and committed while adhering to the critical ACID properties—Atomicity, Consistency, Isolation, and Durability. These properties are essential for maintaining the integrity of data and ensuring the reliability of database systems, especially in environments where data is subject to constant updates, potential system failures, and concurrent access.

This paper provides an in-depth exploration of transaction models in database systems, highlighting their evolution and addressing the key challenges and future directions. Section 1 introduces the importance of transaction models in ensuring consistency, reliability, and performance in modern database systems, outlining the limitations of traditional models in the context of complex applications. Section 2 examines the classical transaction models, including the ACID properties and their application in various database scenarios, discussing their strengths and limitations. Section 3 delves into advanced transaction models, such as nested, distributed, and real-time models, emphasizing their ability to address the challenges of scalability, fault tolerance, and performance in modern systems. Section 4 explores critical issues in transaction models, including deadlock management, distributed constraints, and scalability challenges, while also discussing the trade-offs involved in implementing such models. Section 5 highlights emerging trends and potential areas of future research, including the integration of machine learning, blockchain, and energy-efficient transaction management strategies. Section 6 concludes the paper by summarizing the key findings and identifying the ongoing need for adaptive, innovative transaction models that meet the evolving demands of complex database systems.

II.TRANSACTION MODELS in DATABASE SYSTEMS

This section explores the evolution of transaction models in database systems, highlighting their key principles, strengths, limitations, and applications across various scenarios. The discussion begins with foundational concepts and progresses to specific models, addressing the unique challenges faced by modern database systems.

Database systems play a critical role in managing data efficiently across a wide range of applications, from enterprise solutions to real-time systems. At the heart of database management is the transaction model, which defines how transactions are executed, controlled, and maintained. A transaction consists of a sequence of operations—such as reads, writes, inserts, and deletes—that ensure the consistency and integrity of the database. Effective transaction models are designed to adhere to the ACID properties—Atomicity, Consistency, Isolation, and Durability—which guarantee reliable data management even during system failures or concurrent access. However, traditional flat transaction models often fall short as application requirements become more complex, leading to the development of advanced transaction models. These models modify or extend traditional frameworks to address challenges like distributed databases, nested transactions, long-duration operations, and real-time constraints. Transaction models are essential for defining how transactions are executed and maintained in a system. They help ensure data integrity and consistency in the presence of failures or concurrent operations. The ACID properties, which form the foundation of transaction management, play a crucial role in ensuring reliable transaction behaviour:

A. ***ATOMICITY*** ensures that a transaction is indivisible, either fully completing or not executing at all. If a failure occurs, all changes made by the transaction are undone to maintain consistency.

B. ***CONSISTENCY*** ensures that a transaction moves the database from one valid state to another, adhering to predefined rules like constraints and triggers.

C. ***ISOLATION*** guarantees that concurrent transactions do not interfere with each other, preventing inconsistent intermediate states. Depending on isolation levels, some interactions between transactions are allowed to balance performance and consistency.

D. ***DURABILITY*** ensures that once a transaction is committed, its effects are permanent, even in the event of failures like power outages or system crashes.

Transaction models abstract how transactions are structured and managed. Over time, various transaction models have been developed to meet the needs of more complex applications. For example, in real-world scenarios, transactions may involve long durations or nested operations, such as in Computer-Aided Design (CAD) or real-time systems. Models like the nested transaction model address such complexities, allowing subtransactions to fail without affecting the parent transaction, which improves failure recovery. Similarly, in distributed database systems, where transactions span multiple locations, models like the Flex transaction model relax some ACID properties to enhance scalability and performance in distributed environments.

The following sections introduce several key transaction models, each designed to address specific challenges within database systems, starting with Gray's Model.

A. GRAY'S MODEL

Gray's Model introduced multi-level locking and two-phase commit protocols to enhance data management in distributed systems [9] [10] . This model was designed to support high concurrency and ensure robust recovery mechanisms, making it particularly useful in environments where many transactions occur simultaneously across different sites. The goal of Gray's Model is to maintain data consistency and offer efficient recovery from system failures, even in highly distributed systems. One of its main advantages is its strong support for concurrency and its ability to handle recovery effectively, which is crucial for distributed systems. However, the model introduces complexity, especially with multi-level locking and two-phase commit, which can increase overhead and slow down performance, particularly as the number of transactions increases. The main challenge lies in maintaining consistency and managing the complexity inherent in distributed transaction management.

B. READ/WRITE MODEL

The Read/Write Model treats database objects as pages in memory and defines transactions based on sequences of read and write operations [11]. This model simplifies the design of transactions by focusing on basic read and write actions, making it ideal for simpler transaction environments where complex computations are not needed. The primary goal is to provide a straightforward approach to managing database transactions by focusing on the core data access functions. It offers the advantage of being easy to implement, making it suitable for systems with basic transactional needs. However, it has limitations as it overlooks computations that may occur in main memory, making it unsuitable for applications requiring more advanced processing during transactions. The challenge in this model is its inability to handle complex transactional logic, especially for applications that need more than just basic data manipulations.

C. RELATIONAL UPDATES MODEL

The Relational Updates Model focuses on atomic operations such as insertions, deletions, and updates within a relational database [12]. It was designed to ensure the atomicity and consistency of fundamental relational operations, providing a robust foundation for database transactions that require basic relational data manipulation. The model's goal is to manage database operations in such a way that ensures all transactions are executed atomically, maintaining integrity in the database. Its primary advantage is that it guarantees atomic execution, which is essential for ensuring data consistency during simple database operations. However, it is limited in scope, as it primarily caters to relational databases and doesn't account for more complex transactional requirements. The main challenge is maintaining consistency in concurrent operations, especially when multiple transactions access the same data simultaneously.

D. ONLINE/BATCH TRANSACTIONS MODEL

The Online/Batch Transactions Model categorizes transactions into short-lived (online) and long-lived (batch) transactions, each with different processing needs [13]. Online transactions are designed for quick responses, while batch transactions handle larger volumes of data over longer periods. The goal of this model is to optimize resource usage by tailoring the processing strategy based on the transaction type. Online transactions offer quick response times, making them ideal for real-time applications, while batch transactions allow for more efficient processing of large data sets. However, online transactions are often limited to smaller portions of the database, which may reduce their efficiency when dealing with large datasets. Conversely, batch transactions can require more system resources, especially when handling large amounts of data. The challenge lies in balancing the system's resource usage to optimize both response times for online transactions and the efficiency of batch processing.

E. GENERAL/TWO-STEP/RESTRICTED TWO-STEP MODEL

The General/Two-Step/Restricted Two-Step Model defines transactions through a sequential order of read and write actions, ensuring strict consistency in systems that require specific operational constraints [14] [15] [16]. This model is particularly useful when transactions need to follow a strict sequence of actions to guarantee data integrity. Its primary goal is to maintain consistency by enforcing a structured order of operations during transaction execution. The advantage of this model is its customizability, allowing it to cater to applications with specific transactional order requirements. However, it introduces complexity due to the need for strict sequencing, making it difficult to implement in systems that require flexibility in the order of operations. Additionally, if not carefully managed, the model is prone to deadlocks, especially when multiple transactions require different sequences of operations. The main challenge is ensuring that the transaction flows do not result in deadlocks while maintaining the strict consistency requirements.

F. DISTRIBUTED DBMS MODEL

The Distributed DBMS Model is designed to manage distributed databases across multiple locations, ensuring consistency and fault tolerance in environments where data is spread across different sites [17]. The primary goal of this model is to maintain data consistency and ensure the fault tolerance of the system despite the geographical distribution of the data. The distributed nature of the model allows for scalability, enabling systems to handle larger volumes of data and transactions. However, the implementation of distributed DBMS models is complex due to the need to coordinate transactions across multiple locations, often leading to higher system overhead. The main challenge is ensuring low latency, high consistency, and fault tolerance, particularly in large-scale distributed systems where maintaining consistency and synchronization between multiple nodes becomes increasingly difficult.

G. FLAT TRANSACTION MODEL / ADVANCED TRANSACTION MODEL

The Flat Transaction Model and Advanced Transaction Model cater to different levels of transactional complexity. The Flat Transaction Model handles single-level operations, making it simple and efficient for basic transaction needs [18]. In contrast, the Advanced Transaction Model supports hierarchical transactions, which are more suitable for complex systems that require multi-level or nested operations [19]. The goal of the Flat Transaction Model is to simplify transaction design, while the Advanced Transaction Model aims to efficiently manage more complex transaction flows through hierarchical structures. The advantage of the Flat Model is its simplicity, making it easy to design and implement for basic tasks. On the other hand, the Advanced Model handles complex transactions better, offering improved performance for sophisticated applications. However, the Flat Model is limited to simple operations, making it unsuitable for more intricate applications, while the Advanced Model can introduce higher performance costs and greater complexity, which can impact system efficiency. The main challenge lies in ensuring integrity checks and managing complexity in the Advanced Transaction Model, especially when dealing with nested or hierarchical transaction structures.

In conclusion, Transaction models form the foundation of database systems, ensuring data integrity, reliability, and efficient management across a wide range of applications. From simple flat transactions to complex hierarchical and distributed models, each framework addresses unique operational challenges. As database systems evolve to meet the demands of real-time, distributed, and large-scale environments, choosing the right transaction model is crucial to achieving a balance between performance, fault tolerance, and data integrity. By leveraging tailored models, modern systems can efficiently handle increasingly complex scenarios while maintaining robust functionality.

III. ADVANCED TRANSACTION MODEL in DATABASE SYSTEM

This section examines the evolution of advanced transaction models, outlining their principles, strengths, limitations, and applications. The discussion advances to specific models, highlighting how they address challenges in modern databases by introducing flexibility, scalability, and fault tolerance for complex scenarios.

The evolution of database systems has necessitated the development of advanced transaction models to address the growing complexity and requirements of modern applications. Traditional flat transaction models, although foundational, are limited in their ability to handle intricate scenarios such as long-running processes, collaborative workflows, and distributed systems. As the demand for high-performance, scalable, and resilient database systems has grown, the need for innovative transaction models has become increasingly apparent. Advanced transaction models are designed to overcome these challenges by introducing flexibility and robustness into transaction management. These models are particularly critical in environments where maintaining strict adherence to ACID (Atomicity, Consistency, Isolation, Durability) properties is impractical or counterproductive. By providing mechanisms for concurrency, fault tolerance, and dynamic restructuring, advanced models enable efficient management of complex transactions, often spanning multiple systems or domains. Applications such as Computer-Aided Design (CAD), real-time databases, and workflow systems exemplify the necessity of these models. Advanced transaction models introduce two critical innovations: enhanced operational abstractions and relaxation of the strict ACID (Atomicity, Consistency, Isolation, and Durability) properties. Operational abstractions allow for features like parallelism within transactions, nested transaction hierarchies, and hybrid operations that combine multiple workflows under a unified framework. These structural enhancements improve execution control and enable collaborative workflows. On the other hand, relaxing strict ACID properties—particularly atomicity and isolation—caters to applications requiring cooperative interactions or partial commits for long-running processes. For example, traditional isolation protocols, such as locking-based mechanisms, can hinder shared workflows and increase the likelihood of deadlocks. Gray’s 1981 findings highlighted that the frequency of deadlocks rises exponentially with transaction size, further demonstrating the limitations of traditional protocols in managing large-scale operations. To address these challenges, advanced models such as Nested Transaction Model, Sagas, Multilevel Transaction Model, Dynamic Restructuring, Workflow Models and Flex Transaction Model have been developed. These advanced models ensure resilience, efficiency, and flexibility, meeting the demands of modern applications while balancing the need for consistency and operational fluidity. By supporting long-duration, complex transactions, they provide critical solutions for today's sophisticated and collaborative database applications.

The following sections explore a range of advanced transaction models, each designed to address specific challenges and application needs. These discussions focus on their distinctive features, advantages, and practical uses in modern systems.

The Nested Transaction Model, developed by Moss, is an extension of the flat transaction model, where transactions are organized hierarchically, allowing subtransactions to commit independently [20]. This model improves fault tolerance by allowing subtransactions to fail without impacting the parent transaction, and it enhances concurrency by enabling the parallel processing of subtransactions. Despite its advantages, the model introduces complexity in managing subtransaction states and dependencies, as well as potential issues with maintaining isolation and consistency between subtransactions. It is particularly suited for complex systems requiring high fault tolerance and recovery capabilities, such as enterprise software and real-time systems. The Sagas model, proposed by Garcia-Molina and Salem in 1987, is a variant of nested transactions where each subtransaction is associated with a compensating transaction [21]. If the saga is aborted, compensating transactions are executed to undo the effects of the completed subtransactions. This model is useful in distributed systems with long-running transactions, where it is impractical to lock resources until the entire transaction completes. The primary advantages of sagas are their flexibility in allowing subtransactions to affect the system before the entire saga completes and their applicability in distributed systems. However, they relax isolation between subtransactions, which may violate traditional ACID properties, and introduce complexity in managing compensating transactions. Sagas are widely used in distributed systems involving long-running business processes or workflow systems where partial execution is acceptable. The Multilevel Transaction Model builds upon open nested transactions by organizing subtransactions in a balanced tree structure [22] [23]. This approach aims to improve the efficiency of transaction execution and management by grouping related operations and processing them at different abstraction levels. The model's advantages include better resource management and enhanced performance due to the balanced tree structure, which optimizes execution efficiency. However, the complexity of organizing transactions and managing different levels of abstraction adds overhead to the system. This model is particularly suited for large-scale enterprise applications and systems that require efficient processing of nested transactions at multiple levels of abstraction. The Dynamic Restructuring (Split and Join Transaction Model) introduces flexibility by enabling transactions to be split into multiple smaller transactions or joined together during execution [24]. This model is designed to optimize resource management and parallelism, especially in systems where transactions can benefit from being processed concurrently. The main advantage of this approach is the ability to adapt to changing system conditions and improve parallelism, reducing execution time. However, it increases complexity in managing split and join operations and introduces potential overhead in dynamically restructuring transactions. This model is applicable in real-time systems with fluctuating load conditions and high-performance systems requiring adaptive transaction management. The Workflow Models are designed for complex, long-running transactions where tasks are executed sequentially or in parallel. These models integrate nested transactions and can relax ACID properties to allow for more flexible execution [25]. They are needed in applications with complex task dependencies and dynamic task sequences, such as business process management. The key advantages of workflow models are flexibility in task execution and sequencing, making them suitable for real-world applications with interdependent tasks. However, relaxing ACID properties can lead to potential inconsistencies, and managing dynamic workflows adds complexity. Workflow models are typically used in business process management systems and complex distributed systems, such as e-commerce and supply chain management. Finally, the Flex Transaction Model generalizes ACID properties in multidatabase systems by relaxing atomicity and isolation to provide greater flexibility in distributed and heterogeneous environments [26]. It addresses challenges in systems where full ACID compliance is not feasible or necessary, such as when transaction performance and scalability are prioritized over strict consistency. The advantages of this model include improved performance, flexibility, and concurrency, as well as better scalability in distributed systems. However, relaxing these properties can reduce consistency and introduce complexity in managing the system. The Flex model is most applicable in multidatabase systems and distributed environments, such as large-scale e-commerce platforms or cloud-based applications where high availability is critical.

In conclusion, advanced transaction models play a pivotal role in addressing the limitations of traditional flat transaction systems by introducing innovative mechanisms tailored to the needs of modern applications. Each model offers unique advantages, such as improved concurrency, fault tolerance, and flexibility, while catering to specific use cases ranging from nested hierarchies to distributed workflows and long-running transactions. However, these benefits often come with trade-offs, such as increased complexity and the relaxation of strict ACID properties. By carefully balancing these factors, advanced transaction models provide robust solutions for managing complex, high-performance, and distributed database systems, making them indispensable for contemporary and future applications in diverse domains.

IV.ISSUES AND CHALLENGES in TRANSACTION MODEL

This section discusses the key issues and challenges faced by transaction models in adapting to diverse application requirements. These include scalability limitations, concurrency management, and deadlocks, as well as challenges in fault tolerance, resource management, and execution control.

Transaction models play a pivotal role in maintaining consistency, reliability, and performance in database systems. However, the growing complexity of modern applications, coupled with advancements in distributed, real-time, and large-scale systems, has exposed various limitations and challenges in these models. Traditional transaction models, while robust in ensuring ACID properties, often fall short when faced with the demands of contemporary applications requiring scalability, fault tolerance, and flexibility. Advanced transaction models, though addressing some of these shortcomings, introduce new layers of complexity and performance trade-offs. The Table I below provides a structured overview of these challenges, along with potential resolutions that leverage modern techniques and technologies to address them effectively.

TABLE I TRANSACTION MODEL ISSUES AND REMEDIES

S. No.	Issues in Transaction Models	Description	Issues Resolution
1.	Scalability Limitations	Traditional models struggle with handling large-scale or distributed systems, leading to bottlenecks in processing.	Implement distributed transaction models and load-balancing mechanisms to enhance scalability.
2.	Concurrency Management	Ensuring isolation in concurrent transactions increases resource contention and deadlocks, especially in nested models.	Use advanced concurrency control protocols like Optimistic Concurrency Control (OCC) and Multi-Version Concurrency Control (MVCC).
3.	Deadlocks and Performance Overheads	Locking mechanisms often lead to deadlocks; advanced models may reduce deadlocks but increase dependency management.	Use deadlock detection techniques (e.g., waits-for graph cycles) and lightweight dependency management protocols.
4.	Relaxation of ACID Properties	Relaxing atomicity and isolation introduce inconsistencies and partial failures in transactions.	Use compensating transactions (e.g., Sagas) and hybrid consistency models for balancing flexibility and integrity.
5.	Fault Tolerance and Recovery	Failed subtransactions can disrupt overall	Employ nested transaction models with

		system integrity, requiring complex rollback mechanisms.	independent commit/abort handling and robust recovery protocols.
6.	Resource Management	Models like Sagas and dynamic restructuring demand efficient resource allocation to prevent overloads.	Use resource scheduling algorithms and adaptive resource allocation strategies for dynamic environments.
7.	Heterogeneity in Distributed Systems	Coordinating transactions across diverse data sources and protocols increases complexity.	Implement middleware solutions and consensus protocols like Paxos or Raft for transaction coordination.
8.	Execution Control and Task Sequencing	Dynamic task dependencies in workflows complicate execution control and scheduling.	Use workflow management systems with robust scheduling and dependency tracking mechanisms.
9.	Real-Time Constraints	Maintaining timeliness while ensuring consistency and isolation is challenging in real-time systems.	Use real-time transaction models with deadline-aware scheduling and priority mechanisms.
10.	Security and Privacy	Relaxed isolation raises concerns about secure and private data handling in shared environments.	Adopt encryption protocols, secure access control, and isolation-aware privacy policies.
11.	System Complexity and Overheads	Advanced models introduce significant management complexity in states, compensations, and dynamic operations.	Design streamlined management frameworks with modular approaches to handle complexity in advanced models.

The Table I highlights the diverse challenges encountered in transaction models and offers potential solutions to address them effectively. It underscores the evolving demands on transaction systems, particularly in modern environments like distributed databases, real-time systems, and heterogeneous platforms. Scalability, concurrency, and fault tolerance remain pressing concerns, necessitating innovative approaches like distributed transaction protocols, advanced concurrency controls, and robust recovery mechanisms. Additionally, the relaxation of ACID properties to achieve greater flexibility introduces trade-offs, demanding hybrid consistency models and compensating mechanisms. Real-time constraints, execution control, and task sequencing challenges emphasize the need for deadline-aware models and dynamic workflow management systems to meet stringent performance requirements. Furthermore, issues related to security and privacy in shared environments call for stronger encryption techniques and privacy-aware transaction protocols. While the proposed resolutions aim to mitigate these challenges, it is evident that transaction models must continuously adapt to accommodate emerging technologies and application domains. Research into scalable, adaptive, and energy-efficient transaction models, along with integration into modern frameworks such as IoT, blockchain, and multi-cloud environments, will be pivotal in addressing the future needs of transaction systems. By focusing on these challenges and leveraging advancements in technology, transaction models can evolve to provide robust, efficient, and secure solutions for complex database environments.

V.FUTURE RESEARCH DIRECTIONS

This section outlines future research directions for enhancing transaction models to meet the evolving demands of modern computing systems.

The evolution of transaction models has been driven by the need to address the limitations of traditional approaches and to meet the requirements of increasingly complex database environments. As computing systems continue to expand into areas like distributed systems, real-time applications, and heterogeneous platforms, transaction models must adapt to these emerging challenges. The rapid advancement of technologies such as cloud computing, IoT, and blockchain introduces new demands on transaction systems, including the need for scalability, fault tolerance, real-time responsiveness, and secure processing. Furthermore, the relaxation of traditional ACID properties to accommodate flexibility in collaborative applications adds another layer of complexity to transaction management. Addressing these issues requires innovative solutions and a forward-looking approach to transaction model design. Future research in this domain must explore cutting-edge techniques and methodologies to ensure that transaction models remain robust, efficient, and adaptable to diverse application scenarios. Below is an overview of key future research directions that will shape the development of transaction models.

Future research in transaction models is poised to address the growing complexity and demands of modern computing environments. Scalable transaction models are essential for managing large-scale, distributed, and heterogeneous systems, such as cloud and edge computing, with a focus on lightweight protocols that enhance scalability without sacrificing consistency. Enhanced concurrency control mechanisms are another critical area, involving novel techniques to minimize deadlocks and boost throughput, potentially leveraging machine learning for adaptive lock management and dynamic scheduling. Fault-tolerant systems will benefit from robust recovery mechanisms for nested and distributed transactions, along with proactive fault detection and mitigation strategies for real-time applications. The integration of transaction models with emerging technologies such as blockchain, IoT, and multi-cloud systems will require adaptations to meet unique consistency and performance requirements, with hybrid models combining traditional and blockchain-based management solutions. Relaxation of ACID properties to achieve optimized trade-offs between strict compliance and flexibility is crucial for collaborative and real-time systems, complemented by eventual consistency models with bounded convergence guarantees. Energy-efficient transaction protocols will be vital for sustainable computing, especially in mobile and edge devices. Real-time and predictive transaction management can optimize resource allocation and deadline adherence by anticipating transaction patterns, particularly in time-critical domains like healthcare and finance. Security and privacy enhancements will involve developing secure protocols to protect data integrity and address cryptographic challenges in lightweight, real-time applications. Additionally, flexible transaction models for dynamic workflows can enable seamless task sequencing and execution in evolving systems. Hybrid transaction models, combining the strengths of nested, saga, and multilevel approaches, hold promise for complex and collaborative environments. Establishing standards for transaction protocols will enhance interoperability across diverse platforms, while AI-driven optimization can enable real-time decision-making for execution, conflict resolution, and anomaly detection.

In conclusion, the pursuit of these future research directions represents a transformative opportunity to redefine transaction model capabilities for modern computing. As technology evolves, the demand for scalable, secure, and efficient transaction processing will continue to rise. Researchers and practitioners must collaborate to develop innovative solutions that bridge the gap between theoretical advancements and practical implementation. Moreover, the integration of emerging technologies with transaction models highlights the importance of interdisciplinary research. Collaboration between fields such as artificial intelligence, distributed computing, cybersecurity, and real-time systems will be pivotal in achieving breakthroughs. These advancements will not only address the limitations of current transaction models but also create new possibilities for application in areas like smart cities, autonomous systems, financial technologies, and healthcare informatics. Ultimately, a holistic approach that prioritizes adaptability, resource efficiency, and interoperability will drive the evolution of transaction models. By addressing challenges proactively and embracing cutting-edge techniques, the next generation of transaction models will play a central role in shaping the future of computing systems, ensuring reliability and performance in increasingly dynamic and complex environments.

VI.CONCLUSION

Transaction models have been instrumental in shaping the evolution of database systems, addressing challenges related to reliability, consistency, and scalability in increasingly complex applications. Classical transaction models laid the foundation by ensuring ACID properties, but their limitations in handling modern requirements necessitated the development of advanced models. Advanced transaction models, including nested transactions, sagas, and workflow models, introduced greater flexibility, fault tolerance, and scalability to meet the demands of distributed, real-time, and long-duration applications. These models addressed the trade-offs between consistency and performance while enabling collaborative and dynamic workflows, thus enhancing overall system efficiency. Key challenges in transaction management persist, such as the complexity of distributed and nested transactions, the trade-offs involved in relaxing ACID properties, and the constraints imposed by real-time and large-scale applications. Effective concurrency control, fault tolerance mechanisms, and adaptive transaction protocols are critical to overcoming these challenges and ensuring robust transaction management. Future research directions point to promising advancements, including the integration of machine learning for transaction optimization, blockchain technologies for secure and transparent transactions, and innovative protocols tailored for distributed and real-time environments. These efforts aim to bridge the gap between evolving application needs and the limitations of existing models. This study provides a comprehensive understanding of transaction models, their evolution, and the challenges they address. By analysing their trade-offs and identifying emerging trends, this work underscores the importance of balancing consistency, scalability, and fault tolerance. The continuous evolution of transaction models is vital for meeting the demands of modern applications and ensuring the adaptability and resilience of database systems in the future.

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] C. 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] 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] E. B. Moss, “Nested Transactions: An Approach to Reliable Distributed Computing,” Cambridge, MA, , 1981.

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

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

[23] G. H. C. B. P. a. M. P. Weikum, “Multi-level recovery,” in 9th ACM Symposium on Principles of Database Systems, Nashville, TN, 1990.

[24] C. Pu, “Superdatabases for composition of heterogeneous databases,” in 4th International Conference on Data Engineering, 1988.

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

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

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.