Republic of Iraq

Ministry of Higher Education and Scientific Research

Tikrit University

College of Computer Science and Mathematics

Department of Computer Science

Develop Hybrid Concurrency Cotrol Model for Transaction Processing in Distributed Database System

A Thesis Submitted to

The Council of the College of Computer Science and Mathematics,

Tikrit University,

In Partial Fulfillment of the Requirements for obtaining the Degree of Master in Computer Sciences

By

Muthanna Abdullah Saleh

Supervised by

Asst. Prof. Dr. Saadi Hamad Thalij

2024 A.D. 1446 A.H.

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

﴿ قَالُوا سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَآ ۖ إِنَّكَ أَنتَ الْعَلِيمُ الْحَكِيمُ ﴾

صدق الله العظيم

سورة البقرة : الآية (32)

Acknowledgements

First and formost, all praise be to Allah Almghty for granting me the will

to complete this thesis. Special thanks and appreciation are due to my supervisor

Asst. Prof. Dr. Saadi Hamad Thalij Who supporting and encouraged me in every

step and stage of the thesis to complete it.

Finally, I would like to thank my Parents, my wife, brothers, sisters, my daughters, and everyone who assisted me and

prayed for me to complete this thesis.

SUPERVISOR'S CERTIFICATION

I certify that the thesis entitled " **Develop Hybrid Concurrency Cotrol Model for Transaction Processing in Distributed Database System**" was prepared under my supervision at the Department of Computer Science / College of Computer Science and Mathematics /University of Tikrit is a Fulfillment of the requirement for the degree of Master in Computer Science.

Signature:
Asst. Prof. Dr. Saadi Hamad Thalij
Date: / /2024

REPORT OF DIRECTOR OF POSTGRADUATE STUDIES COMMITTEE

According to the recommendations presented by the supervisors and the linguistic
evaluator of this thesis, I nominate this thesis to be forwarded for discussion.

Signature:
Dr. khalid Khalis ibrahim
Date: / /2024

REPORT OF THE HEAD OF THE DEPARTMENT OF COMPUTER SCIENCE

According to the recommendations presented by the supervisors and the linguistic
evaluator of this thesis and the director of the postgraduate studies committee,
I nominate this thesis to be forwarded for discussion.

Signature:
Asst. Prof. Dr. Mohamed Aktham Ahmed
Date: / /2024

EXAMINATION COMMITTEE CERTIFICATION

We certify that we have read the thesis entitled " **Develop Hybrid Concurrency Cotrol Model for Transaction Processing in Distributed Database System** " and the examining committee examined the student " **Muthanna Abdullah Saleh**" in its contents and what is related with it in / /2024 and that in our opinion it is adequate with ( ) standing as a thesis for the degree of Master of Science in Computer Sciences.

Signature:
Assistant Professor. Dr. Mshari Ayed AL- Askar

Date: / /2024
Chairman

Signature:
Assistant Professor. Dr. Muhaned Thiab Mahdee

Date: / /2024
Member

Signature:
Assistant Professor. Marwa Adeeb Mohammed

Date: / /2024
Member

Signature:
Assistant Professor.Dr. Saadi Hamad Thalij

Date: / /2024
Member and supervisor

Final Approval by the Dean of the College of Computer Science and Mathematics

Asst. Prof. Dr. Mahmood Maher Salih

Date: / / 2024

ABSTRACT

Problem Statement: In the interconnected world of distributed systems, Distributed Database Management Systems (DDBMS) have become critical for managing data spread across multiple locations. DDBMS face challenges in concurrency control, where managing operations from concurrently executing transactions is crucial. Optimistic concurrency control models can suffer from performance issues due to varying conflict rates among transactions. A dynamic approach is needed to adaptively select the best concurrency control strategy based on real-time transaction conditions. **Objective:** The objective of this study is to develop a develop model that dynamically selects between optimistic and pessimistic concurrency control approaches in DDBMS. This study aims to develop an algorithm that dynamically switches between optimistic and pessimistic concurrency control in DDBMS, optimizing performance and consistency by adapting to varying conflict rates. **Methodology:** The proposed algorithm leverages a Back Propagation Neural Network (BPNN) to determine whether to grant a lock or identify the winning transaction in a distributed database environment. The BPNN's parameters are optimized using the Bear Smell Search Algorithm (BSSA), which is designed to fine-tune the network for better decision-making. Recognizing that in practical settings, information about the writeset (WS) and readset (RS) is often only partially available before transaction execution, the algorithm is tailored to accommodate scenarios with optional or incomplete WS and RS data. This dynamic selection mechanism, supported by the BPNN-BSSA combination, allows the system to adaptively choose the best concurrency control strategy based on the current conflict rate. **Results:** The implementation of the BSSA-BPNN algorithm demonstrated significant improvements in both transaction throughput and consistency compared to traditional static concurrency control methods. By dynamically adjusting to the conflict rate in real-time, the algorithm was able to reduce transaction conflicts and improve overall system performance, particularly in high-conflict scenarios. **Conclusion:** The proposed BSSA-BPNN algorithm provides an effective solution for dynamic concurrency control in DDBMS, balancing performance and consistency by adapting to varying conflict rates. The integration of machine learning and optimization techniques into the concurrency control process enhances the system's ability to handle diverse transaction conditions, making it a robust option for modern distributed database

environments. **Limitations and Future Work:** While the BSSA-BPNN algorithm shows promise, it is not without limitations. The effectiveness of the algorithm is contingent on the accuracy of the conflict rate prediction, which can be influenced by unpredictable changes in transaction patterns. Additionally, the computational overhead introduced by the BPNN and BSSA may affect performance in systems with extremely high transaction volumes. Future work should focus on refining the conflict rate prediction mechanism, exploring alternative optimization algorithms, and testing the algorithm in more diverse and larger-scale distributed environments to further enhance its scalability and efficiency.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DDB	Distributed Database
DDBMS	Distributed Database Management System
DBMS	Database Management System
2PC	Two-Phase Commit
3PC	Three-Phase Commit
RDBMS	Relational Database Management Systems
DTP	Distributed Transaction Processing
ACID	Atomicity Consistency Isolation Durability
ERP	Enterprise Resource Planning
DistDGCC	Distributed Dependency Graph Concurrency Control
YCSB	Yahoo! Cloud Serving Benchmark
TPC-C	Transaction Processing Performance Council Benchmark C
ARIES	Aryabhatta Research Institute of Observational Sciences
OCC	Optimistic Concurrency Control
EC2	Amazon Elastic Compute Cloud
S3	Simple Storage Service
DOCC	Deterministic and Optimistic Concurrency Control
R-TBC/RTA	Rose Tree Based Consistency/Rose Tree Algorithm
GPA-BHC	Bayesian hierarchical clustering and Graph Partitioning Algorithm
PEM	Performance, Efficiency, and Manageability
BFD	Byzantine fault detection
FLAC	Failure-Aware Atomic Commit
RLSM	Robustness-Level State Machine

HGP	Hybrid Graph Partitioning
OLB	Optimized Load Balancing
RDMA	Remote Direct Memory Access
OFFEXEC	Optimal Execution Time
OFFCOMM	Competitive Communication
OLTP	Online Transaction Processing
SPTM	Service Proxy Transaction Management
MVCC	Multi-Version Concurrency Control
MSA	Microservice Architectures
HF	Health Factor
ART2	Adaptive Resonance Theory 2
RS	Readset
WS	Writeset
TE	Transaction Effectiveness
HBOCC	Hash-Based Incremental Optimistic Concurrency Control
JAG_TDB_CC	Temporal Database Concurrency Control
SCN	System Change Number
MOCC	Mostly-Optimistic Concurrency Control
MQL	Multi Queuing Lock
CDMS	ClusteriX Data Management System
RDD	Resilient Distributed Dataset
RCN	Record Change Number
MDS	Mobile Database Systems
CHs	Cluster Heads

CCM	Concurrency Control Mechanism
OODB	Object-Oriented Databases
ASOCC	Adaptive and Speculative Optimistic Concurrency Control
2PL	Two-Phase Locking
DCM	Dynamic Concurrency Management
AOCC	Adaptive Optimistic Concurrency Control
DFCL	Deadlock-Free Cell Lock
P-WAL	Parallel Write-Ahead Logging
SAOL	Secondary Asynchronous Optimistic Lock
LRCD	Least Recent Conflict Dependence
TDG	Transaction Dependency Graph
SCT	Scatter-Concurrency Throughput
AdaTAM	Adaptive Variant Thread Activity Management
TAM	Thread Activity Management
ES2PL	Secure 2PL Real-Time Concurrency Control Algorithm
MCSI	Multi-Clock Snapshot Isolation
NVM	Non-Volatile Memory
DPC2-CD	Distributed Processing and Concurrency Control in Cloud Databases
LUN	Logical Unit Numbers
RDM	Raw Device Mapping
NPIV	N_port ID virtualization
DBaaS	Database-as-a-Service
TS-DLP	Time-Stamp-Based Distributed Locking Protocol
RDMA	Remote Direct Memory Access

T/O	Timeout-based Concurrency Control
CLMD	Concurrent Lock Manager For Deterministic Concurrency Control Protocols
MGSA	Modified Gravitational Search Algorithm
OLAP	Online Analytical Processing
FairTID	Fair Thread ID
TEE	Trusted Execution Environment
BPNN	Back Propagation Neural Network
BSSA	Bear Smell Search Algorithm
MP	Master Process
COP	Cohort Processes
UP	Update Processes
I/O	Input/output
C2PL	Conservative two-phase locking protocol
S2PL	Strict two-phase locking protocol
R2PL	Rigorous two-phase locking protocol
T	Transaction
Q	Data Item
TS	Timestamp
POC	Probability Odor Components
POF	Probability Odor Fitness
OF	Odor Fitness
EOF	Expected Odor Fitness
DOC	Distance Odor Components
TP	Throughput

CR	Conflict rate
ET	Execution time
MSE	Mean Squared Error

LIST OF SYMBOLS

S	Signal Transition
R_n	Hidden Layer
$w_{m\,n}$	Weight Connection
θ_n	Threshold Value
v_l	Output of the neuron
EO	Expected Output
$(EO_l - y_l)y_l(1 - y_l)$	Error correction in the output layer
$O_i = [oc_i^1, oc_i^2 \ldots . oc_i^j \ldots oc_i^k]$	odor received with k molecules/components
$OM = [O_i]_{n \times k} = \left[oc_i^j\right]_{n \times k}$	Odor matrix
DS_i^j	Breathing function and glomerular layer process
$MG_i(O_i)$	Olfactory epithelium
$DS = \{ds_1, ds_2, \ldots, ds_n\}$	External Inputs to the Mitral cells
$DS_c = \{ds_{c1}, ds_{c2}, \ldots, ds_{cn}\}$	Central Input to Granule cells
$\varphi_x(X) = \{f_x(x_1), f_x(x_2), \ldots, f_x(x_n)\}$	Outputs of the Mitral cells
$H_0, L_0\ and\ W_0$	Relationship between the Granular and Mitral Cells
$C_l, C_h\ and\ C_w$	Connection Constants
POC	Probability Odor Components
POF	Probability Odor Fitness
OF	Odor Fitness
EOF	Expected Odor Fitness

DOC	Distance Odor Components
$\aleph_1$ and $\aleph_2$	Odors thresholds
$iter_{max}$	Maximum Iterations
E	conflict rate parameter
N_c	number of conflicts
$\alpha, \beta, \gamma \; and \; \delta$	random weight vectors

CHAPTER ONE

INTRODUCTION

It is vital to store data across multiple nodes or locations. The focus was on managing concurrent transactions efficiently and ensuring consistency and isolation properties across the distributed environment. There are different methods based on locking and timestamps, according to which the performance of the proposed model is evaluated in terms of throughput, access time, and scalability. Distributed database systems present challenges that do not exist in centralized systems. Multiple interconnected databases are deployed across different geographical locations to enable efficient data access and management. There are significant challenges due to network latency, data duplication, and node failure. In this research, It developed a hybrid concurrency control model designed to process transactions in distributed database systems. This model must adapt to the dynamic nature of distributed systems, including changing network conditions, node failures, and data evaluation.

1.1. SCOPE AND MOTIVATION OF THE RESEARCH

Investigating concurrency control models specifically tailored for distributed database systems, where data is stored across multiple nodes or sites. Focusing on managing concurrent transactions efficiently and ensuring consistency and isolation properties across the distributed environment is vital. It is also important for exploring a combination of concurrency control mechanisms to leverage the strengths of different approaches, such as locking-based and timestamp-based methods. Assessing the performance of the proposed technique in terms of throughput, latency, and scalability, considering the challenges posed by distributed environments. Addressing various consistency models, such as strict consistency, eventual consistency, and causal consistency, and their implications on concurrency control in distributed settings. Investigating mechanisms to ensure transactional consistency and recoverability in the presence of node failures, network partitions, and other distributed system failures.

Distributed database systems introduce challenges not present in centralized systems, such as network latency, communication overhead, and data partitioning. These challenges motivate the need for tailored concurrency control techniques.

Ensuring that transactions execute concurrently while maintaining data consistency and isolation is crucial for the correctness and performance of distributed systems. Traditional concurrency control techniques may face scalability limitations in distributed environments with a large number of nodes and high transaction volumes. Hybrid techniques offer the potential to improve scalability while maintaining correctness. The motivation to research hybrid concurrency control techniques lies in the desire to optimize the performance of distributed database systems by leveraging the strengths of different concurrency control mechanisms based on the characteristics of the workload and system configuration. Distributed database systems are pervasive in modern applications such as cloud computing, social networks, and e-commerce platforms. Enhancing the concurrency control techniques in these systems can lead to improved reliability, performance, and user experience. Therefore, the research aims to address the challenges inherent in managing concurrency in distributed database systems by proposing and evaluating a hybrid concurrency control technique. This technique is motivated by the need to ensure consistency, scalability, and performance in real-world distributed applications.

1.2. PROBLEM DEFINITION

In a distributed database system, multiple interconnected databases are spread across different geographical locations, enabling efficient data access and management. However, ensuring consistency and concurrency control in such a distributed environment poses significant challenges due to factors like network latency, data replication, and node failures. Concurrency control techniques play a crucial role in maintaining data integrity and transaction correctness in distributed databases. The research problem revolves around developing a hybrid concurrency control technique tailored for transaction processing in distributed database systems. Designing a concurrency control mechanism that ensures transactions can execute concurrently while preserving data consistency across distributed nodes. This involves exploring various concurrency control models, such as locking, timestamp ordering, and optimistic concurrency control, and identifying their limitations in a distributed setting. This model must adapt to the dynamic nature of distributed systems, including varying network conditions, node failures, and data partitioning. The proposed technique should be robust enough to handle these challenges without sacrificing performance or data consistency. It must investigate

optimization strategies to enhance the performance and scalability of the concurrency control technique. This includes exploring techniques to minimize the overhead of coordination among distributed nodes, reduce contention for shared resources, and improve transaction throughput under heavy workloads. These mechanisms must ensure fault tolerance and recovery in case of node failures or network partitions. This involves designing strategies for transaction recovery, data replication, and distributed commit protocols to maintain data consistency and recover from failures seamlessly. This hybrid concurrency control technique must address the unique challenges posed by distributed database systems, offering improved performance, scalability, and fault tolerance while ensuring data consistency and transaction correctness.

1.3. RESEARCH OBJECTIVES

The research advances the system in a distributed environment by integrating Hybrid concurrency control techniques and natural-based search algorithms to effectively enhance efficiency.

- To develop a resilient model to face the failures related to requests, sites, queries, and communication.
- To develop a model that permits parallel execution of transactions and achieves maximum concurrency.
- The computational methods and storage mechanisms should be modest to minimize overhead.
- To develop hybrid concurrency control approach combines optimistic and pessimistic approaches and then dynamically selects the concurrency control technique for the corresponding transaction sets using the conflict rate.

1.4. CONTRIBUTION

A hybrid concurrency control method for transaction processing in distributed database systems is introduced, utilizing artificial neural networks and a bio-inspired optimization algorithm. This hybrid method integrates the Back Propagation Neural Network (BPNN) and Bear Smell Search Algorithm (BSSA). The BPNN method is utilized to handle the lock-grand decisions and to resolve the conflicts by determining the successful transactions. The parameters of the BPNN are fine-tuned by using the natural inspired algorithm BSSA. Regarding the transaction conflicts, resource contention, and system performance metrics the BPNN is trained on the historical data to enable the model to learn the patterns and relationships between various input parameters and for desired concurrency control decisions. In concurrency control, the BPNN is used to handle the complex and non-linear relationships between input parameters and desired outputs. However, the performance of BPNN is highly dependent on the quality of training and optimizing the internal parameters such as weight and bias. To optimize the parameters the BSSA is utilized for making the concurrency control decisions. The proposed technique aims to find the optimal set of BPNN parameters that maximize the accuracy and effectiveness of concurrency control decisions. The main contribution of the model includes:

- The development of a hybrid concurrency control approach combines optimistic and pessimistic approaches and then dynamically selects the concurrency control technique for the corresponding transaction sets using the conflict rate.

- The BPNN determines whether to select the optimistic or pessimistic model to determine the committed or successful transaction.

- The parameters of BPNN are optimized by using the BSSA to enhance the performance and efficiency of the model.

- Evaluations are performed in a small bank model of transaction processing in which the optimistic or pessimistic approach is selected for the customer and internal bank transactions.

1.5. THESIS ORGANIZATION

The thesis chapters are outlined as:

Chapter 1 gives an introduction to the motivation behind the research . It also elaborates on the problem description. The objectives from research and contribution of research .

Chapter 2 gives an introduction to the distributed database management systems are explained in detail. It also includes the concepts, characteristics and types of DDBMS. It also elaborates on the DTP, and ACID properties along with the concurrency control problems, and strategies. The existing research based on the DTP, concurrency control methods and hybrid concurrency control methods along with the inference and limitations are discussed. also discusses the Design of concurrency control in DDBMS. It includes algorithms of distributed concurrency control, the structure of a distributed transaction, and modeling of DDBMS and concurrency control protocols for the DDBMS model.

Chapter 3 gives an overview of the methodology of the hybrid concurrency control. The methodology includes transaction management, back propagation neural network (BPNN) and bear smell search algorithm (BSSA). The integrated technique BPNN- BSSA for the hybrid concurrency control model is also explained in detail along with the implementation of the model in this chapter.

Chapter 4 discusses and analyses the results that are obtained from the model. It also includes the experimental setup along with the evaluation metrics. The performance results are analyzed in this chapter.

Chapter 5 encompasses a summary discussion, conclusion, and the conclusive results derived from the research. Additionally, it integrates potential findings for future research, aligning with current demands and requirements.

1.6. CHAPTER SUMMARY

In this chapter, the management of concurrent transactions and their properties of consistency and isolation across a distributed environment are briefly explained, with a focus on hybrid control methods and a discussion of the models and methodologies that have been developed. Recent and current research on distributed transaction processing and concurrent control methods has been extensively reviewed

CHAPTER 2

DISTRIBUTED DATABASE MANAGEMENT SYSTEMS AND DESIGN OF CONCURRENCY CONTROL IN DDBMS

INTRODUCTION

The Distributed Database Management Systems have gained a vital importance in this interconnected world. It is important to understand the characteristics, architecture and various aspects to effectively manage and process data in a distributed environment (Diallo et al., 2013). This chapter provides a detailed introduction to the various concepts and gives an in-depth exploration of each aspect. This chapter discusses the characteristics, types, and ACID properties along with transaction processing and management. This research thoroughly illustrates the objectives and motivation behind its methodologies.

This research analyzed hybrid concurrency control techniques which integrated optimistic and pessimistic methods for transactions in a distributed database (DDB) system. The primary objective of the work is to effectively enhance the efficiency and performance of the model. This research also helps in managing the transaction and versioning mechanisms which ensure and enhance the concurrency control. The research integrates neural network and optimization techniques to address the challenges associated with concurrent access to shared resources in a database system. The Hybrid Concurrency Control Technique helps in selecting the concurrency approaches and the neural network determines to grant lock for winning transactions and finally optimization technique to optimize the parameters for better performance of the model with increased efficiency.

Design of Concurrency Control refers to the design principles and strategies involved in implementing concurrency control mechanisms within a DDBMS. Concurrency control ensures that multiple users or transactions can access and modify shared data simultaneously without causing inconsistencies or violating integrity constraints. In DDBMS, where the database is distributed across multiple nodes in a network, designing effective concurrency control mechanisms is crucial. This involves selecting appropriate algorithms like locking or Timestamp ordering, defining transaction management protocols to maintain ACID properties, addressing issues of replication and consistency through techniques such as distributed locking and commit protocols, specifying isolation levels, implementing deadlock detection and resolution mechanisms, optimizing

performance to minimize overhead, and tackling challenges such as communication overhead, data distribution, fault tolerance, and scalability

2.1. DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

Due to the advanced technologies in database and storage, the company has a concentrated database at one mainframe or the computer server connected to the internet for worldwide access. This method has a disadvantage in that if anyone's site is blocked or goes down, all the connected sites also get blocked from accessing the data from the database until the overall system gets back up. To address and avoid these problems a centralized database also known as a distributed database (DDB) is created. The DDB is a collection of interconnected databases which are distributed across a computer network. Distributed Database Management System (DDBMS) is a software system which is created to manage, maintain and control databases which are physically distributed across multiple locations. The DDBMS manages the DDB and provides the access to the user (Dinh et al., 2018). The objective of DDBMS is to deliver transparent access to data along with data integrity, consistency and availability.

The data are distributed in multiple locations which are influenced by factors like geographical dispersion of organization, data replication for fault tolerance and requirements for performance. DDBMS is used to handle and analyze data within the distributed settings. The database also provides a transparent and unified view of data, which allows the users and applications to access, maintain and manipulate the data which are stored in a single and centralized database. The DDBMS integrates various methods and techniques such as data fragmentation, replication, distributed transaction management, concurrency management control and query processing in distributed systems (Gharaibeh et al., 2017). The methods are used to enhance and ensure the integrity and consistency of data. This is also used to handle limitations in distributed environments namely network failures, site failures and concurrent access to data.

2.1.1. Characteristics of Distributed Databases

The DDB has several Characteristics which are used to understand, implement and manage the DDBMS (Coronel et al., 2019). The main characteristics are as follows:

1. Data Distribution: the data is physically dissipated across various sites or nodes within a computer network rather than centralized as a single database. The data distribution means that the data is partitioned and stored in different locations.
2. Data Replication: The data can be duplicated (replicated) across various sites which improves the scalability, availability and reliability. This method also ensures if one site fails or gets down, the data can be accessed from other sites.
3. Fragmentation: The data in DDB can be partitioned into smaller units or fragmented called fragments. These fragments can be either horizontally or vertically distributed across the systems or sites based on the performance, usage patterns and security requirements.
4. Transparency: this refers to the ability to hide the complexities of the data distribution and fragmentation details from the users and application.
5. Heterogeneity: the database can be heterogeneous, which can integrate different types of database management systems (DBMS) and data models. This method allows the users to leverage the existing models and technologies.
6. Distributed Transaction Processing: To maintain the ACID properties across multiple sites, the transactions in a DDB require a specialized protocol like a two-phase commit (2PC) or a three-phase commit (3PC), this processing is called distributed processing.
7. Fault Tolerance: The DDB is designed to handle failures like site failures, network failures or communication failures. To ensure fault tolerance techniques like data replication, failover mechanisms and recovery procedures are developed.
8. Performance and Scalability: By processing and distributing the data across multiple sites, DDB can enhance and improve performance and scalability by leveraging parallel processing and load-balancing techniques.
9. Data Communication cost: Effective management of data communication costs involves optimizing network resources, minimizing data transfer, ensuring efficient communication protocols and prioritising critical data.

10. Reliability: Reliability refers to the DDB consistently and accurately storing, retrieving and processing data despite various failures. High reliability in DDB involves implementing robust fault-tolerance mechanisms and maintaining redundant copies of data.

2.1.2. DDBMS Architecture

The architecture of DDBMS comprises various components and layers to manage and process the data across the distributed environment. There are various methods for dividing the functionality among various processes related to DDBMS (Abdelhafiz et al., 2020). The architecture of DDBMS includes Client/server systems, Collaborating server systems and middleware systems.

CLIENT/SERVER SYSTEMS:

A client/server system has one or more servers and one or more client processes, in which the user can send a request or a query to any server for processing. The user interfaces or applications that interact with the DDBMS for performing data-related operations such as updating, querying and retrieving information are processed in the client system. Database server manages retrieval, data storage and processing tasks. The server systems also handle the request from the client, coordinate communication with other components and execute database operations in DDBMS architecture (Özsu et al., 1999). The Client/server system consists of a number of clients and a few servers connected to the network. The client can send a request/query to any one server. This architecture is easy to implement and executes the process.

Figure 2.1 represents the architecture of a Client/server with two clients and a server connected through a network.

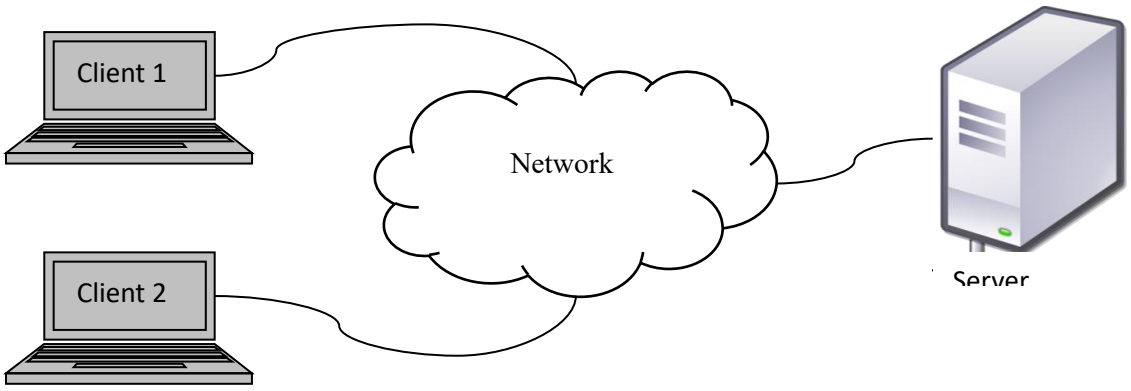


Figure 2.1 Client/server architecture

There are three different client/server architectural approaches, they are:

1. Two-Tier Client/Server Architecture
2. Three-Tier Client/Server Architecture
3. Multi-Tier Client/Server Architecture

2.1.2.1 Two-Tier Client/Server Architecture:

In the two-tier architecture, the model is classified into two main components such as the client and the server. The two-tier architecture process is similar to the Client/server application where the client manages the application's logic and user interface and the server manages the queries and processing tasks. The main features of this architecture include that the model is easy to implement (Ali et al., 2020). Since the model has direct communication between the client and the server the performance of the system is also enhanced.

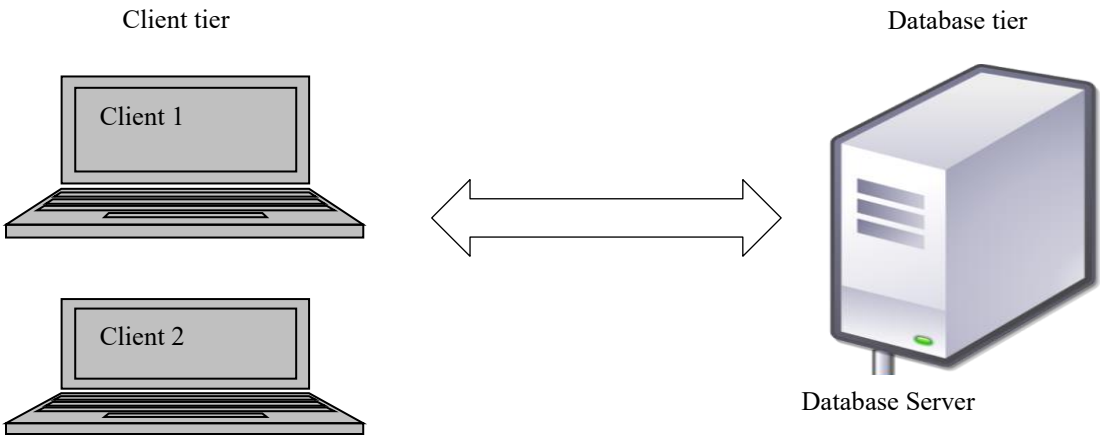


Figure 2.2 Two-Tier Client/Server Architecture

Figure 2.2 represents the Two-Tier Client/Server Architecture, which includes the Client Tier and Database Tier

2.1.2.2 Three-Tier Client/Server Architecture:

In the three-tier architecture, a middleware or the application server layer is integrated between the server and the client layer. The middleware is considered to be the intermediary between the client and the server which helps in handling tasks including communication management, task processing and task execution. The main advantage of the middleware layers is that the layer can distribute client requests across multiple servers which enables load balancing and enhances resource utilization (Abba et al., 2020). Figure 2.3 illustrates the architecture of a Three-tier client/ server.

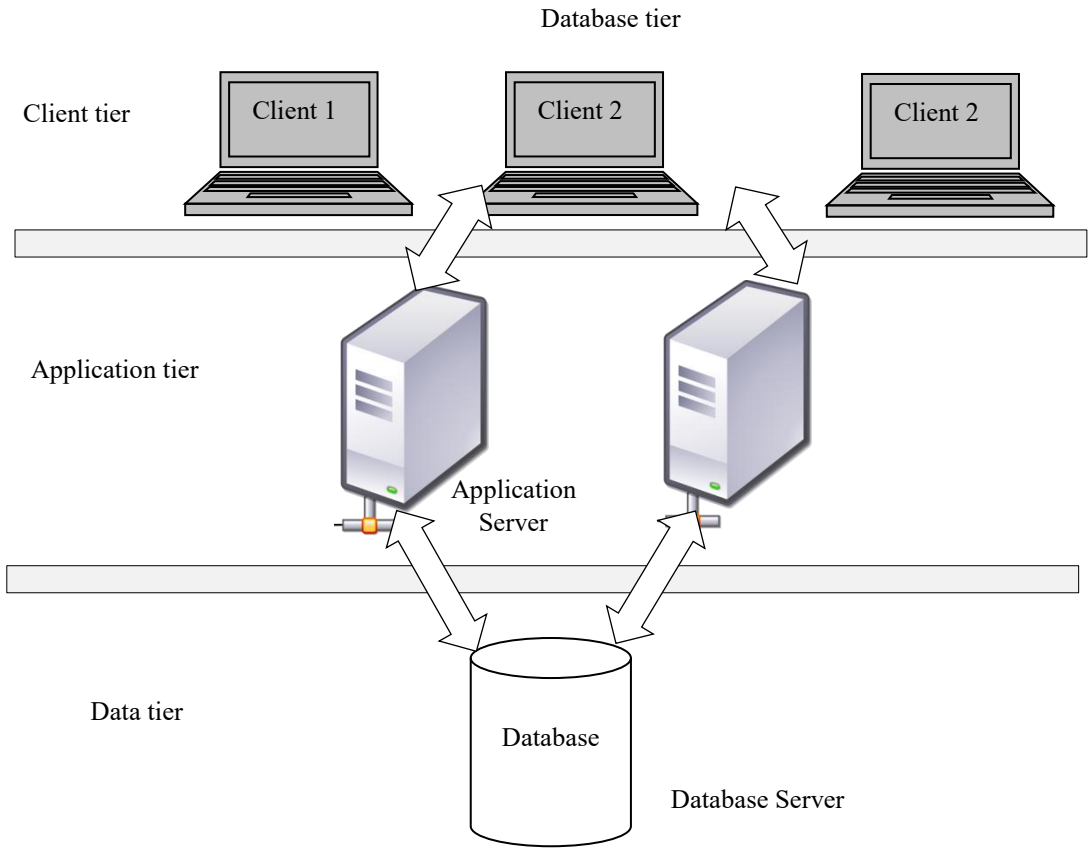


Figure 2.3 Three-Tier Client/Server Architecture

2.1.2.3Multi-Tier Client/Server Architecture:

The multi-tier architecture includes adding additional layers or tiers to the model to accommodate the complex requirements of the application to execute the process. The additional layers may include cache servers, security servers and messaging servers (Olivares-Rojas et al., 2020). This model provides a flexible and scalable framework to improve the performance and to support diverse computing environments.

2.1.3. Types of DDBMS

In distributed systems, data is spread across various database systems within an organization. These database systems are interconnected through communication links, facilitating access to the data for end-users. The DDBMS types are classified as,

1. Homogenous DDBMS
2. Heterogeneous DDBM

1.Homogenous DDBMS:

Homogenous database systems operate on the same operating system, utilize the same application processes, and employ similar hardware devices. Each site/query runs the same type of DDBMS, ensuring uniformity across the distributed environment (Mahajan et al., 2023). Homogeneous DDBMS encompass Autonomous and Non-Autonomous subtypes.

i. Autonomous DDB: In Autonomous DDB, Each DDBMS operates autonomously, exchanging messages to facilitate the sharing of data updates.

ii. Non-Autonomous DDB: In Non-Autonomous DDB, a primary DDBMS manages database access and updates across the nodes.

2. Heterogeneous DDBMS:

Heterogeneous distributed database systems typically operate across diverse operating systems, employ varying application procedures, and utilize different hardware devices. Each site/query may run in different DDBMSs, including various relational database management systems (RDBMSs) or even non-relational DBMSs (Zhang et al., 2023). This means that different DDBMSs are employed at each node within the system. Heterogeneous DDBMS encompass Systems and Gateways as subtypes.

i. Systems : This provides support for either some or all of the functions within a single logical database. Systems are further classified as Full DBMS functionality and Partial-multi database.

1.Full DBMS functionality: It encompasses all functionalities typically found in a distributed database.

2.Partial-multi database: It provides support for certain functionalities of a distributed database. Partial-multi databases are further classified into federated Distributed Database and unfederated Distributed Database.

3.Federated Distributed Database: Facilitates the utilization of local databases to fulfill specific data requirements. This can be again classified as loose integration and tight integration.

a. Loose integration: In loose integration, various schemas are present for each local database, requiring interaction between each local DBMS and all local schemas.

b. Tight integration: A global schema exists and outlines the entire data encompassing all local databases.

4. Unfederated Distributed Database: All access needs to be routed via a centrally located coordinating module.

ii. Gateways: direct connections of simple paths are created to other databases, by lacking the advantages of one logical database. The types of distributed DDBMS are depicted in Figure 2.4.

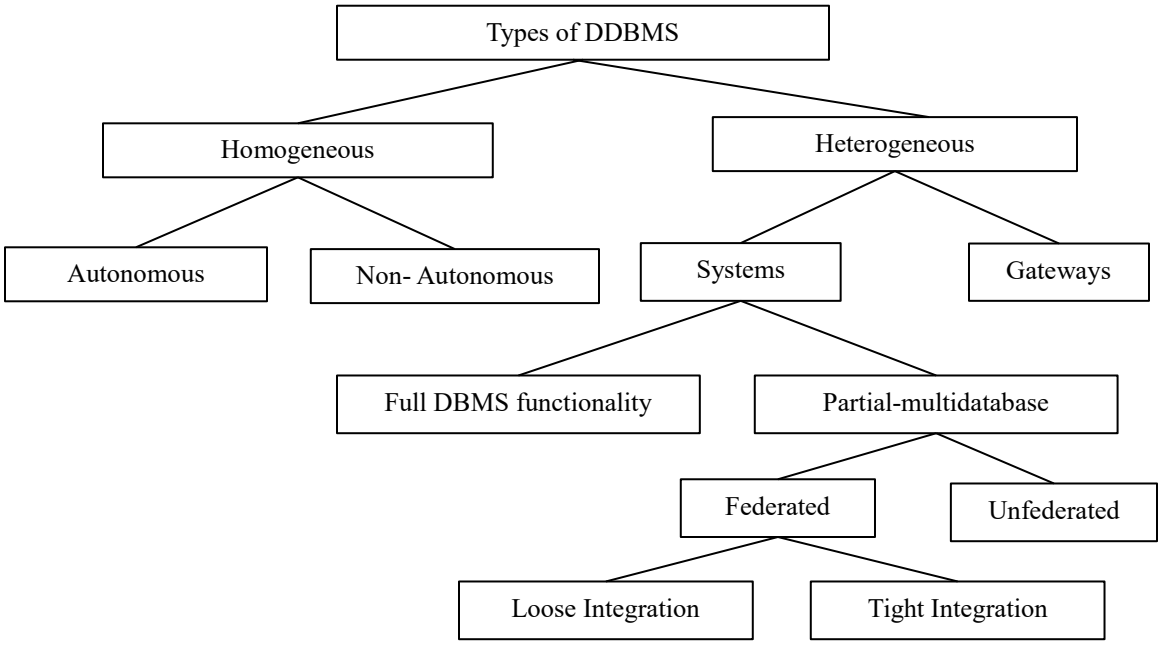


Figure 2.4 Types of DDBMS

2.1.4. Distributed Transaction Processing (DTP)

DTP refers to the management and coordination of transactions that span multiple nodes, sites or queries in a distributed computing environment. A transaction refers to a logical unit which consists of one or more database operations including write, read or update. In distributed systems, a transaction involves modifying and accessing data that are stored in different databases across different locations across the network (Taft et al., 2014). The components in DTP include,

1.Transaction Manager: This is liable for correlating the execution of distributed transactions. This method also ensures that transactions follow ACID properties. The transaction manager initiates and monitors transactions, manages the state information of the transactions and coordinates with the resource manager present at each node.

2.Resource manager: The resources including databases, files and services are managed by the resources manager in the distributed transactions. Each resource manager is connected to the local database or resources, which are responsible for executing transactional operations on those resources. This also ensures that the transactional changes are committed or rolled back consistently across all resources.

3.Concurrency Control: Concurrency controls are employed to manage concurrent access to shared data by multiple transactions. This method includes techniques like locking and timestamp ordering which prevents data inconsistencies and ensures that the transactions are done consistently (Dickerson et al., 2017).

4.Deadlock Detection and Resolution: DTP systems employ deadlock detection algorithms to identify and resolve deadlocks that occur when transactions wait indefinitely for resources held by other transactions. The technique for deadlock resolution includes techniques like deadlock detection with timeouts or deadlock prevention strategies. This method also helps to mitigate the impact of deadlocks on system performance. DTP enables applications to perform complex transactional operations across the distributed environment along with data integrity, consistency and reliability in the distributed computing environments.

2.1.5. ACID (Atomicity, Consistency, Isolation, Durability) Properties of Transactions

The ACID properties are the set of four characteristics which ensure the reliability and consistency of transactions in database systems.

Atomicity: atomicity ensures that a transaction is treated as a single logical unit of work. Either all of the transactions are completed successfully or all the transactions are aborted. If any of the systems fail due to an error and system crash, the entire transaction is rolled back to its initial state (Lotfy et al., 2016). This atomicity property ensures the consistent state of the database.

Consistency: this is considered to be a vital property which ensures the consistent state for each database instance. This also assures that the database remains in a valid state before or after the execution. Each transaction must maintain the data integrity, data validity and constraints that are given by the schema (Xia et al., 2016). If any transaction ignores the rules or constraints, the transaction will be aborted and the database will be reverted to the previous consistent state.

Isolation: The isolation property guarantees that transactions execute independently of each other without interference. Each transaction operates in isolation from others until the completion of the transaction. This also assures that the intermediate results of the transactions are not visible or accessible to other transactions. Isolation levels such as read committed, read uncommitted and repeatable read provide different levels of isolation to control the visibility and accessibility of data among the transactions.

Durability: Durability ensures that the committed transactions persist even when the system fails or crashes. Once the transactions are completed and committed successfully the changes are saved permanently which cannot be lost due to hardware failures, system crashes or power failures (Chittayasothorn 2022). These changes are typically recorded in non-volatile storage to ensure durability.

The ACID properties ensure that the database transactions are executed reliably, and consistently which enhances the data integrity and isolation even in the system failure or errors. This helps to build robust and transactional consistent database systems.

2.1.6.Transaction Management applications

Transaction management applications in DDBMS involve handling transactions that span multiple nodes or sites within a distributed computing environment (Muslihah et al., 2020). The common transaction management applications in DDBMS include,

1. **E-commerce Platforms:** E-commerce Platforms facilitate transactions between buyers and sellers over the Internet. The transaction details such as placing orders, inventory updates, payment processing and generating invoices involve interacting with a distributed database that stores product details and transaction records.
2. **Supply Chain Management:** Supply Chain Management correlates the flow of goods, information and finances across a distributed network between the suppliers and manufacturers. The transaction details like processing and management require access to DDB at different points in the supply chain.
3. **Enterprise Resource Planning (ERP):** An ERP system combines multiple business functions such as finance, human resources, manufacturing, and sales into a unified platform. Transactions in ERP systems involve activities such as processing purchase orders, generating invoices, recording payroll, and managing inventory, which may extend to multiple departments or locations within an organization.

Transaction management mechanisms such as distributed concurrency control, distributed deadlock detection, and distributed recovery protocols are employed to coordinate and manage transactions effectively across distributed databases.

2.1.7.Concurrency Control Problem

A database system operates with high concurrency, and increases the conflicts, particularly when two concurrent transactions attempt to update the same data simultaneously. If only one transaction were allowed to execute at any given time, operations would proceed sequentially without issue (Harding et al., 2017). However, challenges arise when multiple transactions are updated to the same database item concurrently, while still upholding data consistency. The concurrency problem arises due to the following factors,

1. **Data Inconsistency:** Concurrent execution of transactions can lead to data inconsistency if transactions read and write data simultaneously without

proper coordination. For example, if two transactions read the same data concurrently and perform conflicting updates, this results in an inconsistent state of the database.

2. **Lost Updates:** When several transactions endeavour to modify the same data simultaneously, loss of updates can occur, where one transaction's changes overwrite another transaction's changes without being properly reflected. This can lead to the loss of valuable data modifications and compromise data integrity.
3. **Uncommitted Data:** Transactions may read intermediate or uncommitted data produced by other transactions, leading to inconsistent or incorrect results. This phenomenon, known as dirty reads, this situation arises when a transaction accesses data that has been altered by another transaction but hasn't been finalized or committed.
4. **Concurrency Anomalies**: Concurrent execution of transactions can result in various concurrency anomalies such as dirty reads, non-repeatable reads, and phantom reads. These anomalies occur due to the interleaving of transaction operations and can lead to unexpected or incorrect query results.
5. **Deadlocks:** Concurrent transactions may compete for resources, leading to deadlock situations where transactions wait indefinitely for resources held by other transactions (Neumann et al., 2015). Deadlocks can result in system performance degradation and deadlock resolution overhead.

2.1.8.Concurrency Control Strategies

Concurrency control strategies are techniques employed in database management systems to ensure that transactions execute in a controlled manner, minimizing the risk of data inconsistency and preserving data integrity in a multi-user environment. This method also ensures that transactions execute safely and efficiently in a multi-user environment (Al-Qerem et al., 2020). The concurrency control strategy depends upon factors such as system requirements, transaction workloads and characteristics along with the performance. The concurrency control strategies are classified into:

1. Optimistic Concurrency Control
2. Pessimistic Concurrency Control
3. Hybrid Concurrency Control

Optimistic Concurrency Control (OCC): allows transactions to execute without initially obtaining locks. Instead, it detects conflicts at the time of transaction commit. Transactions record their read and write sets during execution (Jepsen et al., 2018). At commit time, conflicts are checked, and the transaction is either committed or aborted based on the presence of conflicts.

The optimistic approach defers the checking of concurrency constraints, such as ensuring transaction isolation and other integrity rules like serializability and recoverability, until the transaction's completion. This method avoids blocking transactional operations and only aborts the transaction if it violates the desired rules upon committing. When a transaction is aborted, it is promptly restarted and re-executed, which does entail some overhead. However, in a multiuser transaction environment where relatively few transactions are aborted, the optimistic mechanism proves to be effective in achieving its intended purpose. Optimistic concurrency control is well-suited for environments where conflicts are infrequent, such as read-heavy workloads or scenarios with low contention for resources (Bernstein et al., 2015). It avoids the overhead associated with locking and can lead to improved throughput and reduced contention. Optimistic concurrency control offers a balance between ensuring data consistency and maximizing concurrency, making it a valuable strategy in many database systems, particularly those with high read/write ratios.

Pessimistic Concurrency Control (PCC): PCC is a strategy used in database management systems to ensure transaction isolation and maintain data consistency by pessimistically assuming that conflicts will occur and taking preemptive measures to prevent the conflicts. Pessimistic Concurrency Control provides a conservative approach to concurrency management. While it ensures transaction isolation and data consistency, it can lead to increased contention and reduced concurrency, especially in environments with high transaction rates.

Pessimistic concurrency control is executed via locking. Once a transaction acquires a lock on a data object, it remains locked, preventing other transactions waiting for the same data object from accessing it. Only when the transaction that initially obtained the lock releases it can other transactions acquire the lock on the same data object. In the pessimistic locking method, data scheduled for updates is preemptively locked. Once locked, the application can proceed with the modifications, and then either commit or rollback the transaction, which automatically releases the lock (Chaudhry et al., 2022). If another transaction tries

to obtain a lock during this process on the same data, it will be compelled to wait/queue until the first transaction concludes. This method involves acquiring a write lock on the object, executing its operations, copying the updates, and subsequently releasing the lock on the object. This approach operates on the assumption that another transaction may alter the data between the read and the update. To avoid such changes and data inconsistency, the read statement secures the data, preventing any alteration by other transactions.

Hybrid Concurrency Control: Hybrid concurrency control techniques have emerged as a promising solution for transaction processing in distributed database systems. These techniques aim to integrate the benefits of optimistic and pessimistic concurrency control strategies. The hybrid approach is the adaptive concurrency control, which dynamically adjusts its concurrency control strategy based on the current workload characteristics and system conditions. This monitors various parameters such as transaction arrival rates, data contention levels, and system resource utilization (Stephan et al., 2017). Based on these parameters, it adapts its concurrency control strategy, employing either optimistic or pessimistic techniques as appropriate. This adaptive approach aims to provide the best performance and concurrency levels by tailoring the concurrency control mechanism to the prevailing workload conditions.

1. **Pessimistic Component:** In the hybrid concurrency control approach, specific transactions may utilize a pessimistic locking strategy to ensure access to critical data. Pessimistic locking is typically used for transactions that involve highly sensitive or frequently accessed data, where the risk of conflicts and contention is high (Wu et al., 2017). Pessimistic concurrency control maintains transaction isolation reduces the chances of conflicts by obtaining locks on data items in advance, and prevents concurrent transactions from accessing or modifying the same data simultaneously.

2. **Optimistic Component:** Optimistic concurrency control is often employed for transactions that involve less critical or less frequently accessed data, where conflicts are less likely to occur. Instead of acquiring locks preemptively, optimistic concurrency control mechanisms rely on conflict detection and resolution mechanisms to address conflicts at the time of transaction commitment (Wei et al., 2018). If conflicts are detected during the commit, the transaction may be aborted and restarted to ensure data consistency.

3. **Dynamic Selection:** The decision to use pessimistic or optimistic concurrency control for a particular transaction may be dynamically determined based on factors such as transaction characteristics, data access patterns, and system workload. Transaction scheduling algorithms or policies may be employed to analyze transaction requirements and select the most appropriate concurrency control strategy for each transaction dynamically.

For example, transactions accessing critical or highly contended data may be assigned pessimistic locking, while transactions accessing less critical or less contended data may be assigned optimistic concurrency control.

The main advantage of Hybrid concurrency control is flexibility and adaptability which allows database administrators or developers to tailor concurrency control strategies to the specific needs and characteristics of their applications and environments. This method integrates the strengths of pessimistic and optimistic concurrency control, the hybrid approach can provide improved performance, scalability, and resource utilization in database systems with diverse transaction workloads and access patterns.

A BENEFITS AND LIMITATIONS OF HYBRID CONCURRENCY CONTROL MODEL FOR TRANSACTION PROCESSING

BENEFITS	LIMITATIONS
1. It combines benefits of both optimistic and pessimistic methods to optimize performance and ensure data consistency	1. Integrating the Multiple concurrency control mechanism without increasing the complexity of the system is the primary challenge in hybrid concurrency control.
2. Transactions are initially executed optimistically, allowing them to proceed without acquiring locks	2. Coordinating various approaches in a highly concurrent environment increases the synchronization overhead and degrades the consistency
3. the system monitors for potential conflicts using techniques such as validation checks or versioning mechanisms	3. The model adds to space complexity by storing multiple types of information, including transaction status and details, within the system.
4. . If conflicts are detected, the system switches to a pessimistic mode, where transactions acquire locks on data items	4. exhibit limitations such as high abort rates, long execution times, and scalability issues.

Table 2.1 benefits and limitations of hybrid concurrency control model

2.2. LITERATURE REVIEW

The increasing application and effectiveness of the Hybrid Concurrency Control Technique for Transaction Processing in Distributed Database Systems have propelled the IT industry towards a modern computing model. While Hybrid Concurrency Control has a huge volume of potential, it also has a lot of challenges to overcome. This chapter discusses the recent research works on Distributed Transaction Processing, Concurrency Control Methods and Hybrid Concurrency Control Methods to upgrade the distributed database environment.

2.2.1. Distributed Transaction Processing

This section of the research model concentrates on conducting a comprehensive examination of different Distributed Transaction Processing methods as well as identifying the limitations present within the system.

1. Yao et al., (2018) presented a novel concurrency control protocol called Distributed Dependency Graph Concurrency Control (DistDGCC) and a new logging technique called Dependency Logging for distributed in-memory OLTP systems. DistDGCC processed transactions in batches resolved dependencies and built dependency graphs before execution, reducing aborts and thread blocking. Dependency Logging exploited the dependency graphs constructed by DistDGCC for efficient logging and parallel recovery. Two variants are proposed, fine-grained and coarse-grained Dependency Logging, trading off between log size and recovery parallelism. DistDGCC constructed dependency graphs in parallel, resolving intra- and inter-transaction dependencies for local and distributed transactions. Dependency Logging transforms these graphs into log records with dependency information. Experiments on YCSB and TPC-C showed that DistDGCC outperformed 2PL and OCC in throughput and latency with distributed transactions and dependency Logging achieved higher runtime performance and faster recovery than ARIES logging. Yet, the model lacked balance to log granularity and recovery parallelism.

2. Lokhande et al., (2019) proposed an innovative method for ensuring secure, energy-efficient, and scalable transaction management within heterogeneous distributed real-time replicated database systems. This model incorporates several components, including workload characterization utilizing K-means clustering, resource allocation facilitated by the provision of virtual machines, and the implementation of security measures through encryption and decryption of user data. The k-means clustering algorithm is used for workload characterization and task classification, and Amazon AWS services like EC2, DynamoDB (cloud database), and S3 are utilized for resource allocation and data storage. AES algorithm is used for encryption and decryption of user data to ensure security. The performance of the proposed system is evaluated

based on parameters like throughput, CPU utilization, energy efficiency, scheduling time, and response time. The model provided secure data storage and transaction management in cloud environments. Yet, the model lacked a clustering technique for workload characteristics.

3. Dong et al., (2020) presented a new concurrency control protocol called Deterministic and Optimistic Concurrency Control (DOCC) for deterministic databases. DOCC allowed the optimistic execution of transactions and enforced determinism lazily, improving scalability within a single node. It used multi-version snapshots to prevent read-only transactions from blocking execution and employed data prefetching to speed up transaction retries. The model evaluated DOCC using the TPC-C on an 8-node cluster and showed that DOCC outperformed the popular deterministic database Calvin by 4.31x to 8.46x under low and high contention scenarios, respectively. DOCC achieved better performance by exploiting parallelism opportunities and reduced overhead from deterministic locking. The advantages of DOCC include improved scalability, better performance under contention, and increased generality. However, the paper does not address the potential scalability issue of the sequencer component in a larger distributed setting.

4. Dizdarevic et al., (2021) presented a novel approach called Rose Tree Based Consistency/Rose Tree Algorithm (R-TBC/RTA) for managing consistency in highly-distributed transactional databases in a hybrid cloud environment. The model proposed a visible adaptive consistency model and the R-TBC/RTA approach based on Bayesian hierarchical clustering and Graph Partitioning Algorithm (GPA-BHC) for intelligent partitioning of the cloud database. Constructing the R-TBC tree using the Rose Tree Algorithm (RTA) and distributing database partitions across the tree structure based on PEM (Performance, Efficiency, and Manageability) metrics. Experimental validation of the R-TBC/RTA approach demonstrated improved response time, reduced inconsistency window, and better load balancing. The study is limited to a specific scenario for energy sector companies.

5. Yamada et al., (2022) introduced Scalar DL, a middleware designed for transactional database systems to detect Byzantine faults (BFD). Scalar DL enabled the concurrent execution of non-conflicting transactions while maintaining strict serializability. The process involves three phases ordering, committing, and validation, which occur between a primary and secondary server situated in distinct administrative domains. In comparative evaluations, Scalar DL surpassed the other BFD approach by a factor ranging from 3.5 to 10.6 times in terms of throughput. Scalar DL exhibited effective performance across various database implementations. Moreover, it demonstrated near-linear scalability of 91% with an increase in the number of nodes composing each replica. However, it's worth noting that Scalar DL is limited to supporting only two administrative domains.

6. Ali et al., (2022) presented a practical framework for creating a Proxy Database to prevent direct access to distributed transaction databases and ensure information security. The proposed approach entails establishing a Proxy Database comprising a collection of proxy tables and views sourced from various relational distributed databases operating across different domains within an organization. The model created proxy tables that provide roundabout access to data in distributed tables, maintaining location transparency, and mapping proxy tables with local distributed tables and views. It submitted queries on proxy tables, which are decomposed into sub-queries and executed on respective local databases, with results composed and returned. The model handled local database structure differences and propagating changes to the Proxy Database to maintain consistency and implemented a software firewall for network security, with the Proxy Database Server and local databases behind the firewall. The advantages include improved security, location transparency, and efficient data retrieval through a single query across multiple databases. However, the document does not provide a detailed analysis of query processing, optimization techniques, or performance evaluation.

7. Diop et al., (2022) proposed a DDSampling algorithm for randomly drawing patterns from a transactional distributed database, with probability proportional to an interestingness measure combining

frequency and length-based utility functions. The model utilized a decentralized approach which avoids centralizing the entire database by just centralizing the lengths of transaction parts from each fragment and the pre-processing phase computes a weight matrix M containing the lengths of each transaction in each fragment. The pattern drawing phase used M to randomly draw a transaction identifier based on weight, and a length based on weight and then samples a pattern of length from the transaction identifier. The experiments evaluated DDSampling on datasets, showed its lower communication cost compared to centralization, and its robustness against node failures. Yet, the model only analysed communication cost, without detailed results on computational efficiency.

8. Shen et al., (2022) presented DrTM+B, a dynamic live reconfiguration method designed for distributed transaction processing systems. DrTM+B utilized a pre-copy-based mechanism to minimize performance disruption during live reconfiguration processes. It leveraged common features found in modern transactional systems, such as data replication for fault tolerance, DrTM+B and reduced data transfer and downtime. The system employed a cooperative commit protocol to synchronize data and state among replicas, ensuring consistency. Additionally, DrTM+B introduced a hybrid copy scheme that combines pre-copy and post-copy approaches to enhance performance during data migration. Evaluation of a functional system based on DrTM+R with 3-way replication, using TPC-C and SmallBank workloads, indicates that DrTM+B experiences only minor performance degradation during live reconfiguration while maintaining high availability. Furthermore, reconfiguration time and downtime are kept minimal.

9. Pan et al., (2023) presented a novel and practical atomic commit protocol called Failure-Aware Atomic Commit (FLAC) for distributed transaction processing. The model used a robustness-level state machine (RLSM) to monitor the operating environment and dynamically switch to the most suitable protocol. RLSM's parameters are fine-tuned by reinforcement learning to adapt to unstable environments. The experimental results showed that FLAC achieved up to 2.22x throughput improvement and

2.82x latency speedup compared to existing protocols in high-contention workloads. The limitations include higher message complexity when transactions involve many participants and potential liveness.

10. Bharati et al., (2023) presented a novel hybrid graph partitioning and optimized load-balancing approach for scalable distributed transactions. Hybrid Graph Partitioning (HGP) combines vertical and graph partitioning to effectively partition data based on related content. The optimized Load Balancing (OLB) approach calculated the weight factor, average workload, and partition efficiency and balanced the load across partitions. The proposed approach is evaluated using the TPC-E OLTP benchmark dataset. Performance metrics like throughput, response time, distributed transactions, and CPU utilization are analysed. The results showed significant improvement over existing approaches like schema-level partitioning, scalable partitioning, and graph partitioning. Effective data partitioning with related data items improved transaction processing and optimized load balancing which enhanced performance for increasing workloads. Yet, the model lacked handling skewed data distributions.

11. Zhang et al., (2023) proposed RedT, a novel distributed transaction processing protocol for heterogeneous networks RDMA within data centres and TCP/IP across data centres. The model used a pre-write-log mechanism and reduced inter-data-centre round-trips from 6 to 2 and extended sub-transactions from replica granularity to data centre granularity, leveraging RDMA for intra-data-centre communication. The model evaluated on YCSB and TPC-C benchmarks showed RedT and achieved up to 1.57x higher throughput and 0.56x low latency. However, RDMA currently limited to local area networks, may not scale to wide area networks.

12. Poudel et al., (2024) introduced offline algorithms such as OFFEXEC, designed for optimal execution time, and OFFCOMM, aimed at achieving 2-competitive communication costs. The model also proposed both partial and fully dynamic algorithms to optimize both execution time and communication costs within distributed transactional memory systems.

These algorithms were developed to provide efficient solutions for ordered scheduling. The efficacy of these algorithms was assessed through rigorous analysis and validation via various benchmarks applied to random and grid graphs. The model evaluated the ordered scheduling challenges inherent in committing transactions based on predefined priorities within the control-flow distributed transactional memory model. However, the model concentrates on the control-flow model and the results may not extend to other models.

13. Weng et al., (2024) presented Lauca, a workload replicator that generates synthetic workloads for benchmarking transactional database performance. The model introduced Transaction Logic, Data Access Distribution, and Partition Access Distribution to characterize the runtime workload attributes of OLTP applications. The model also proposed algorithms to synthesize workloads and data, ensuring similarity in execution semantics between synthetic and real application workloads. Transaction Logic captured transaction structure and parameter dependencies, data Access Distribution models skewed, dynamic, and continuous data access patterns and partition Access Distribution controls distributed transaction ratios for distributed databases. Extensive experiments on centralized (MySQL) and distributed (TiDB, OceanBase) databases showed that Lauca consistently generated high-quality synthetic workloads. Yet, the model Parameter dependencies in Transaction Logic are not exhaustively defined.

14. Kniep et al., (2024) introduced Pilotfish, the first distributed execution engine for blockchains that can scale out by leveraging multiple machines and executionWorkers under the control of each validator node. The methods utilized are a protocol to efficiently fetch transactions from SequencingWorkers and dispatch them to appropriate ExecutionWorkers. An efficient memory-storage interface to handle partial crash recovery across workers A novel versioned-queue scheduling algorithm to support dynamic reads/writes while allowing concurrent execution. The results showed that for compute-intensive workloads, Pilotfish achieved linear scalability in throughput as more ExecutionWorkers were added, reaching

up to 8x higher throughput. The model limitations include slightly sublinear scalability for non-compute-bound workloads.

15. Nikolić et al., (2024) presented the Service Proxy Transaction Management (SPTM) approach for providing scalable reads and ACID transactions in microservice architectures (MSA). The model used inbound messages to services for transaction management, making it transparent to services and also utilized Multi-Version Concurrency Control (MVCC) with timestamp ordering for concurrency control. The model implemented a transaction manager called fed-agent based on the SPTM approach. The results showed that the fed agent outperforms 2PC by up to a factor of 2 in low-medium contention scenarios and provided high consistency. However, limitations include performance degradation in high contention scenarios and the potential for a high number of aborted transactions.

2.2.2. Concurrency Control Methods

In this part, the research model focuses on concurrency control methods and limitations that exist in the system.

1. Sheikhan et al., (2013) presented a neural-based concurrency control (NCC) method for database systems. The approach utilized the Adaptive Resonance Theory 2 (ART2) neural network to compute a health factor (HF) for transactions, aiding in determining the winning transaction when accessing database objects. Additionally, it factored in the optional knowledge of readset (RS) and writeset (WS) before transaction execution. Training the ART2 neural network involved transaction features such as RS, WS, and user priority. HF was derived based on the transaction and abort counts within each cluster generated by ART2, while the NCC algorithm utilized HF or transaction effectiveness (TE) to select the winning transaction. As a result, the model yielded 1954 aborts and an execution time of 46745 seconds for 1000 transactions. The proposed model demonstrated enhanced concurrency and performance by notably reducing aborts. Yet the model performance improvement depends on transaction rate and RS/WS knowledge and considers only serializability criteria, not other correctness criteria.

2. Ramesh et al., (2015) presented a Hash-Based Incremental Optimistic Concurrency Control (HBOCC) algorithm for distributed databases. The approach implemented optimistic concurrency control within the distributed database model. It addressed transaction conflicts by employing the MD5 hashing algorithm to compute hash values for data items before write operations. By comparing the current and previous hash values, the model detected modifications made by concurrent transactions, aiming to prevent unnecessary transaction restarts. This was achieved by updating hash values and retrying the transaction. The model yielded an execution time of 151 seconds for 1000 transactions. Findings indicated that the HBOCC model exhibited reduced space complexity and demonstrated superior performance, particularly with larger transaction volumes. Limitations lacked exploring different hashing algorithms, handling hash collisions, and addressing security concerns in distributed environments.

3. Neumann et al., (2015) presented a novel multi-version concurrency control (MVCC) implementation for main-memory database systems. The model enabled efficient execution of both transactional and analytical workloads while maintaining serializability guarantees. The method involved updating data in-place, storing versions as before-image deltas, and using precision locking for serializability validation and retained high scan performance using VersionedPositions synopses. The model result analysis showed that the MVCC model has overhead compared to single-version systems, even with serializability. The model outperformed existing MVCC implementations and scales well for both OLTP and OLAP workloads. However, limitations include false positive aborts for minimum/maximum aggregates due to the lack of predicate logging for them.

4. Gohil et al., (2016) presented a research study on the development, implementation, and performance evaluation of a concurrency control algorithm called JAG_TDB_CC, tailored for temporal databases. The proposed algorithm combined the advantages of both timestamp and locking approaches for concurrency control and utilized Oracle 12c's temporal validity support and efficient locking mechanism. The algorithm

prevented conflicting user sessions from acquiring locked resources rather than blocking preventing indefinite waiting times. It incorporated a System Change Number (SCN) as an extra concurrency parameter for condition validation. The results showed a significant reduction in session waiting times when compared to both pessimistic and optimistic concurrency control techniques. However, the algorithm's application is not tested in time-critical real-time database applications.

5. Wang et al., (2016) presented a novel concurrency control scheme called Mostly-Optimistic Concurrency Control (MOCC) and enhanced the performance of Optimistic Concurrency Control (OCC) for extremely competed dynamic workloads on modern servers with thousands of CPU cores. MOCC judiciously used pessimistic locking for suitable records to prevent clobbered reads in scenarios with high conflict levels, while retaining OCC's high performance for low-conflict workloads. MOCC also introduced a cancellable reader-writer spinlock, MOCC Multi Queuing Lock (MQL), which scales well on deep memory hierarchies supports various locking modes and dynamically adjusts itself to optimize performance for workloads with varying contention levels and access patterns. Experiments on a 288-core server showed that MOCC outperformed OCC and pessimistic locking by 8x and 23x, respectively, for high conflict YCSB workloads and achieved 17 million TPS for TPC-C and over 110 million TPS for YCSB without conflicts, 170x faster than pessimistic methods. Yet, the MOCC lacked performance in distributed or partitioned workloads.

6. Choi et al., (2016) presented a concurrency control method to provide transactional processing for cloud data management systems (CDMSs). The proposed method is based on Spark, a distributed processing framework that operates in-memory the system employs the Resilient Distributed Dataset (RDD) model to ensure fault tolerance. The method loads the database from the CDMS into distributed main memory as an RDD and manages versions of the RDD using multi-version concurrency control. Evaluation results showed that the method scales well with the number of transactions. The benefits of the proposed approach are its ability to be seamlessly incorporated into current CDMSs without modification and its scalability. Yet, the model lacks in overhead of

managing multiple versions of RDDs, which may impact performance for large datasets.

7. Gohil et al., (2016) presented a study on the implementation of optimistic locking and concurrency control for temporal databases using Oracle 12c Enterprise Manager. The study creates temporal tables in Oracle 12c and implements an optimistic locking mechanism using a trigger. The trigger used a Record Change Number (RCN) to control concurrent updates to the same row. Multiple user sessions are simulated to perform concurrent updates, and the behaviour is observed graphically using Oracle Enterprise Manager. The optimistic locking approach allowed conflicting transactions to proceed without blocking each other, reducing the risk of deadlocks. The session waiting time for conflicting transactions is decreased compared to pessimistic locking. The graphical representation illustrated the concurrency situations, locking, and unlocking scenarios effectively. The study is limited to Oracle 12c and may not be directly applicable to other database systems.

8. Mohana et al., (2017) presented a hierarchical replication and multiversion concurrency control model for mobile database systems (MDS). The methods involve estimating node distance, cluster formation, replication, mobility prediction, and concurrency control using version numbers and timestamps. The model chooses cluster heads (CHs) by considering their distance relative to past node movements to handle mobility and disconnections and replicates data from the server to nearby CHs on demand or proactively. The model maintained a concurrency table at each CH for each replicated data object. The results showed that the proposed approach reduced energy consumption and overhead compared to a concurrency control mechanism (CCM). It achieved 26% less response time, 25% higher throughput, and 76% lower server load without mobility. With mobility, it achieves 28% less response time, 16% higher throughput, and 64% lower server load.

9. Jun et al., (2018) presented a concurrency control scheme for object-oriented databases (OODBs). OODBs have complex modeling capabilities compared to relational databases, necessitating more intricate concurrency control mechanisms that can impact overall performance. The proposed

approach addresses the concept of class hierarchy within OODBs and operates on an implicit locking scheme. It is specifically designed to utilise data access frequency to mitigate locking overhead in comparison to implicit locking methods. In cases where access frequency information is unavailable, the scheme seamlessly operates similarly to existing implicit locking mechanisms. The results demonstrated that the proposed scheme outperformed implicit locking. The value of the results lied in providing a more efficient concurrency control mechanism for OODBs, potentially enhancing overall system performance. Limitations may include the need for access frequency information and the complexity of handling class hierarchies.

10. Liu et al., (2018) proposed an Adaptive and Speculative Optimistic Concurrency Control (ASOCC) protocol for efficient distributed transaction processing. The model dynamically adapts between OCC and two-phase locking (2PL) based on data access frequency, and speculatively restarting transactions early based on correlated data accesses. The method classified data as cold (OCC), hot (2PL), and warm (speculative restart) based on access patterns. It embedded 2PL into OCC for hot data, performs validation and restart for warm data correlated with hot data. The speculation strategy saved CPU cycles wasted on transactions destined to abort. Performance evaluation on TPC-C workloads showed the speculative restart provided up to 35% throughput gain for workloads with medium data correlation levels and moderate distributed transactions, outperforming priority-based speculation. However, the gain diminishes with excessively distributed transactions.

11. Wang et al., (2018) built a model, concurrency-aware model that determined nearly optimal soft resource allocation for each tier by integrating operational queuing laws with detailed online measurement data. A dynamic concurrency management (DCM) framework was developed, seamlessly integrating the concurrency-aware model to dynamically reallocate soft resources during system scaling. The model compared DCM with Amazon EC2-AutoScale using six real-world bursty workload traces revealing DCM's prowess in significantly reducing latency and increased throughput under all workload traces. During system scaling the work highlighted the significance of coordinating

software and hardware resources. The proposed DCM framework provides an intelligent approach to managing concurrency and improved performance during scaling. Limitations include the overhead of the DCM framework and its impact on system performance.

12. Ding et al., (2018) presented a framework for enhancing the performance of OLTP systems based on OCC utilizing grouping transactions into batches and rearranging operations. The model used storage batching to reorder transactions performed reads and writes operations at the storage layer to reduce conflicts involving identical objects and utilized validator batching for reordering transactions before validation to reduce conflicts between transactions. Efficient algorithms for transaction reordering based on various precedence policies, including reducing tail latency and thread-aware policies for multi-threaded architectures. Experiments are evaluated on a research prototype, an open-source OLTP system, and a production OLTP system demonstrated that the proposed techniques can increase transaction throughput by up to 2.2x and reduce tail latency by up to 71% on workloads with high data contention. The proposed framework leveraged the unique opportunities for batching in OCC systems and provides a holistic approach to improving performance through transaction batching and reordering. It enables the use of OCC with higher-contention workloads by reducing conflicts and improving throughput and latency.

13. Guo et al., (2019) presented an Adaptive Optimistic Concurrency Control (AOCC) for heterogeneous workloads in in-memory database systems. The model identified the high cost of validating operations in transactions, especially for key-range scan operations and under varying workloads. AOCC adaptively chooses low-cost validation schemes based on the operation types and workload characteristics at runtime and assigns validation methods at the transaction level based on operation features, and for each operation, it selects the validation method based on the number of accessed records and workload characteristics. The method is evaluated through experiments, showing good performance and scalability under heterogeneous workloads with point accesses and predicate queries. The model doesn't include specific validation schemes or the adaptation mechanisms used in AOCC.

14. Mohammad et al., (2019) proposed an improved algorithm called Deadlock-Free Cell Lock (DFCL) for database concurrency control. A DFCL implements cell-level locking and prioritizes transactions based on the number of executed statements when addressing conflicts. Additionally, it utilizes triggers to identify and resolve deadlocks by aborting the transaction with the least number of executed statements. Through simulations, DFCL achieved an average of 1234 committed transactions, 266 rolled-back transactions, and an execution time of 3 seconds. The benefits of DFCL include enhanced concurrency, commit rate, throughput, response time, and elimination of deadlocks. However, it increases space complexity due to storing the status of each cell and transaction details.

15. Nakamura et al., (2019) presented a study on integrating the TicToc concurrency control protocol with the parallel write-ahead logging (P-WAL) protocol. The integration of TicToc with P-WAL, a parallel logging protocol designed for non-volatile memory, was employed in the model. Four protocols were developed by combining TicToc with P-WAL, each with or without ELR and group commit functionalities. These protocols were implemented and assessed on multicore systems. The research offered significant insights into the obstacles associated with merging concurrency control and logging protocols. However, it is limited by the use of a simplified database system and storage latency emulation.

16. Al-Qerem et al., (2020) introduced a new optimistic concurrency control protocol variant for cloud-fog environments to reduce communication overhead to the cloud server. The proposed augmented partial validation protocol allows read-only IoT transactions to be processed locally at the fog node, while only update transactions requiring final validation are sent to the cloud. Update transactions go through partial validation at the fog node first, increasing their likelihood of committing at the cloud. This reduced cloud computational and communication while supporting transactional service scalability for IoT applications. Experiments compared the protocols with and without fog node partial validation deployment. Results showed reductions in miss rate, restart rate and communication delay across all three protocols when using partial validation. Communication delay was significantly reduced, enabling fog

computing services with low latency to IoT applications that prioritize minimizing delays. Limitations include the need for caching to compensate for the abort rates introduced by optimistic concurrency control.

17. Fan et al., (2020) proposed 2PC*, a novel concurrency control protocol for distributed transactions across multiple microservice protocols. The model introduced a Secondary Asynchronous Optimistic Lock (SAOL) to replace the less efficient synchronous blocking lock, thus reducing overhead stemming from transaction surges. Additionally, it employed a runtime protocol based on directed graphs to streamline and reorder transaction conflicts. Techniques such as TLA+ were utilized for protocol correctness verification, while Least Recent Conflict Dependence (LRCD) was adopted to simplify transaction conflicts. Furthermore, a middleware implementation was developed using Spring-Boot and Netty. As a result, the model achieved a response time of 623.2ms and processed transactions at a rate of 304.6 per second. The outcomes demonstrated that 2PC* yielded higher throughput, with 67% lower latency and a superior commit rate. However, the model failed to manage unsuccessful transactions.

18. Jin et al., (2021) proposed a novel two-phase concurrency control protocol to optimize the execution of smart contracts in permissioned blockchain systems. The main node processed transactions concurrently through a batching OCC protocol that involved transaction reordering. By employing a greedy algorithm, the system tackled the Min-Density FVS problem to create a transaction dependency graph (TDG) characterized by significant parallelism. A graph partitioning algorithm then segmented the TDG into sub-graphs, effectively minimizing communication overhead while maintaining parallelism. Outcomes demonstrated notable speed enhancements for the primary validators, coupled with a substantial 90% decrease in communication overhead. The model limitations include the NP-hardness of the optimization problems.

19. Liu et al., (2022) proposed a Scatter-Concurrency throughput (SCT) model and identified the near-optimal resource allocation for each server. The model also integrated concurrency awareness and enhanced the

reallocation of soft resources. The experiments are conducted for web applications like EC2-autoscaling and Kubernetes HP. The model reduced the response fluctuations time in the container-based environments.

20. Masumura et al., (2022) proposed a new concurrency control method called Thread Activity Management (TAM) and its adaptive variant (AdaTAM) for in-memory database systems. TAM regulated the number of active worker threads and reduced contention and cache misses. AdaTAM dynamically adjusted the number of active threads based on throughput. The model hypothesizes that reducing the number of active threads is the primary reason for performance improvement, rather than reducing re-conflicting events. Experiments showed that Silo+TAM outperformed under high-contention workloads and AdaTAM automatically converged to the optimal number of active threads, making it suitable for dynamic workloads. The paper does not explore applying TAM to other concurrency control protocols.

21. Abduljalil et al., (2022) presented a new secure two-phase locking (2PL) real-time concurrency control algorithm (ES2PL) for multilevel secure distributed database systems. The proposed ES2PL algorithm introduces a secure manager responsible for determining the security level and handling transaction processing based on real or virtual locks and version data. This method effectively mitigates transaction miss rates, deadlock occurrences, unnecessary aborts, and the starvation of long-running transactions. Simulation results indicate that ES2PL outperforms existing algorithms such as S2PL, E2PL-HP, and RACE in terms of transaction miss percentage, rollback frequency, and average number of committed transactions. Specifically, the model achieves an average of 930 committed transactions, 180 rolled-back transactions, and an execution time of 3 seconds. The benefits include enhanced concurrency, throughput, response time, and overall database performance.

22. Liu et al., (2022) presented Multi-Clock Snapshot Isolation (MCSI), a concurrency control mechanism for implementing isolation snapshots in a non-volatile memory single-layer (NVM) database. The model used multi-version storage to guarantee both durability and runtime access while maintaining only a single copy of data and employed an efficient

generation of snapshots with vector clocks achieved through multi-clock transaction timestamp assignment. The method used per-thread transaction status arrays in NVM and identified uncommitted versions, avoiding the need for centralized components like lock managers. Results showed that MCSI's maximum transaction throughput is 101-195% higher than PostgreSQL-style concurrency control for the YCSB workloads, and 25-49% higher for TPC-C workloads. Transaction latency remains stable as thread count increases, the MCSI model provided 86.2μs being 65-84% lower than the baseline for YCSB and 16-43% lower for TPC-C with 18 threads. However, the work is limited to snapshot isolation level only.

23. Di Sanzo et al., (2022) developed an analytical model to estimate the performance effects of different data access patterns, including random access, ordered access by all transactions, ordered access by a percentage of transactions, and reverse-ordered access. The model's accuracy is validated through extensive simulations, achieving a mean absolute percentage error of less than 2.2% in the worst case. The findings are demonstrated to have practical applications in improving the performance of common transactional benchmarks by modifying the data access patterns. However, the limitations are the focus on the ETL protocol and the assumption of eager transaction aborts.

24. Xia et al., (2023) proposed a deterministic concurrency control approach to efficiently execute blockchain transactions in parallel. The approach divides transactions into batches to mitigate inter-batch conflicts, thereby lessening the burden on proposers. Introducing a two-stage method known as DVC aims to establish a partial order with significant parallelism. DVC demonstrated an enhancement in speed of up to 5X, showcasing how data skewness and transaction cost distributions impact the efficacy of various strategies. However, the performance of DVC depends on the effectiveness of the RedLS algorithm for vertex colouring.

25. Yadav et al., (2023) presented an architecture called Distributed Processing and Concurrency Control in Cloud Databases (DPC2-CD) for distributed processing and concurrency control in cloud databases over a public cloud environment. The architecture comprised clustered ESX

servers, raw disks, storage arrays, and raw Logical Unit Numbers (LUNs). To bolster hardware-level security, the architecture incorporated port zoning, LUN masking, Raw Device Mapping (RDM), and N_port ID virtualization (NPIV). Additionally, it implemented Database-as-a-Service (DBaaS) for application-level security via data encryption. The algorithms encompassed transaction decomposition based on data items, transaction allocation to ESX servers with workload balancing, and a time-stamp-based distributed locking protocol (TS-DLP) for concurrency control and the release of locks upon commit. The model yielded an energy savings boost of 20-25%. Nonetheless, it exhibited performance limitations in dynamic environments.

26. Jingyao et al., (2023) presented an RDMA (Remote Direct Memory Access) optimization technique for two-phase locking (2PL) concurrency control in distributed databasesThe approach involved enhancing and streamlining the lock acquisition and release processes of NO WAIT and WAIT DIE 2PL algorithms using RDMA one-sided primitives. This included introducing strategies like one-sided mutex lock and one-sided mutex/shared lock. The research found that the one-sided mutex lock strategy excelled in scenarios with minimal contention, whereas the one-sided mutex/shared lock strategy performed better in situations with high contention. Experimental findings from the YCSB benchmark demonstrated that the optimized NO WAIT and WAIT DIE algorithms achieved significant performance enhancements, with improvements of up to 5.3x and 10.6x observed under high contention workloads. However, the limitations include the complexity of implementing complex operations using RDMA one-sided primitives and the potential performance impact of retries due to atomicity violations.

27. Zhao et al., (2023) proposed RCBench, an RDMA-enabled transaction framework that aimed to achieve transaction scalability over high-performance RDMA networks. The approach involved abstracting six concurrency control primitives, namely ReadD, WriteD, AtomicD, ReadT, WriteT, and AtomicT, which leverage one-sided RDMA (Remote Direct Memory Access) verbs for efficient remote data access. Additionally, the model introduced optimization principles aimed at reducing RDMA verb invocations during the re-implementation of various concurrency control

algorithms such as Two-Phase Locking (2PL), Optimistic Concurrency Control (OCC), Timeout-based Concurrency Control (T/O), Multi-Version Concurrency Control (MVCC), and Silo. The outcomes demonstrated that RCBench achieved significant performance enhancements, with up to a 42X improvement over TCP/IP, by capitalizing on the capabilities offered by RDMA. However, it is limited to static transactions and may require changes for dynamic workloads.

28. Uchida et al., (2024) proposed a novel concurrent lock manager (CLMD) for deterministic concurrency control protocols. CLMD enabled the concurrent execution of non-conflicting transactions by assessing conflicts between the current transaction and all subsequent transactions. It managed lock requests without obstructing transactions and processed non-conflicting transactions simultaneously. Findings demonstrated a throughput increase of up to 44.4 times compared to traditional Calvin when utilizing 64 logical cores, with transaction execution time averaging 1.49 seconds. Notably, CLMD eradicated the bottleneck caused by unnecessary blocking in conventional lock managers. However, the performance difference between CLMD and conventional methods shrinks under low contention.

2.2.3. Hybrid Concurrency Control Methods

1. Sheikhan et al., (2013) proposed a novel hybrid optimistic/pessimistic concurrency control (HCC) algorithm for centralized database systems using a modified gravitational search algorithm (MGSA)-optimized adaptive resonance theory (ART) neural model. The conflict rate of the transactions is selected based on the optimistic and pessimistic methods. The MGSA-ART model used the ART2 neural network with parameters optimized by MGSA and calculated the health factor (HF) of transactions for comparing and detecting the winner transaction. The proposed HCC algorithm was evaluated in a simulated banking environment with different transaction rates. The model resulted in 1049 number of aborts and an execution time of 37872s for 1000 transactions in pessimistic mode and in the optimistic model number of aborts of 1328 and an execution time of 36248s for 1000 transactions. The results showed that when the transaction rate is 1000 per second, the algorithm achieved a 35% reduction in the number of aborts.

The proposed model improved the concurrency and reduced aborts. The model increased computational complexity.

2. Moiz et al., (2015) presented a hybrid concurrency control strategy for mobile database systems. The proposed hybrid strategy combined the benefits of pessimistic and optimistic approaches by using priority-based locking. In this model, transactions are assigned priorities, initially set to zero and executed optimistically, tolerating conflicts, with priorities increased upon conflicts. Once a transaction reaches the maximum priority, it acquires a lock and executes pessimistically for a limited time. After execution, it releases the lock and reverts to optimistic mode. Simulation results for an m-banking application scenario demonstrate improved throughput, reduced waiting time, and a gradual decrease in starved transaction requests compared to pure pessimistic or optimistic strategies. Yet, the model lacked in determining appropriate maximum priority values for different transaction types.

3. Yu et al., (2018) presented Sundial, a distributed concurrency control protocol that addressed two major limitations of distributed transactions such as high latency and high abort rates. The model dynamically determines the sequencing of transactions during runtime determined by their data access patterns using logical leases, which reduces transaction abort rates. It allowed the database to cache remote data in the local main memory maintain cache coherence and minimize the overhead associated with accessing remote data. The model is implemented in Sundial in a distributed DBMS testbed and the results showed that Sundial outperformed the protocol that follows is outperformed by as much as 57% under conditions of high contention and boosts performance by up to 4.6 times in scenarios with significant access skew due to caching. However, it has limitations such as extra storage overhead for maintaining logical leases and potential performance degradation for partitionable workloads.

4. Gabriel et al., (2020) presented a hybridized concurrency control technique that integrated two-phase locking and timestamp ordering for optimizing transaction processing in distributed database systems. The hybridized method proposed in the study improved performance by enabling certain

operations that would typically conflict in a strict two-phase locking scenario, all while upholding data consistency. This technique involves assigning global timestamps to transactions and permitting concurrent write operations on the same data item by monitoring lock timestamps. Experimental findings demonstrated that the hybridized approach completed transactions approximately 30 seconds faster on average, resulting in reduced execution times overall. The model lacked exploring factors like system load in more complex scenarios.

5. Žužek et al., (2020) proposed an agile development methodologies like Scrum have seen widespread adoption in software development due to benefits like increased flexibility, faster time-to-market, and improved productivity. Researchers have attempted to apply agile principles to physical product development as well, creating hybrid models that combine agile methods with traditional stage-gate processes. The proposed hybrid maintains the overlapping stage structure but uses Scrum for day-to-day work to increase agility. This facilitated faster requirements clarification and change response. Potential benefits include improved efficiency, stakeholder satisfaction, and competitive advantage through increased innovativeness and flexibility. Limitations include the need for empirical validation, analyzing project fit, addressing distributed team challenges, and determining the ideal scope for applying Scrum across development loops.

6. Wang et al., (2021) presented RCC, a unified and comprehensive RDMA-enabled distributed transaction processing framework (RDMA-RCC) that supports six serializable concurrency control protocols. RCC enabled an impartial and equitable comparison of the protocols conducted within a standardized execution environment, where the concurrency control protocol serves as the sole variable component. The model analysed stage-wise latency breakdowns and developed effective hybrid execution which leveraged both RDMA's two-sided and one-sided primitives for different protocol stages. Hybrid implementations outperformed pure two-sided or one-sided counterparts by up to 67% in throughput. However, the model increased the complexity of the system load.

7. Lyu et al., (2021) present an extension to the Greenplum database system to enable it to handle both Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) workloads. Greenplum was traditionally an OLAP data warehouse with limited OLTP capabilities. The model identified bottlenecks like inefficient locking mechanisms and the two-phase commit protocol that hindered OLTP performance. The methods utilized a global deadlock detector to increase query concurrency, a one-phase commit protocol when transactions update data on a single segment, and a resource group model to separate OLTP and OLAP workloads for more suitable query processing. The evaluation using TPC-B and CH-benchmark shows improved OLTP performance without sacrificing OLAP capabilities. However, the model lacks detailed performance comparisons against specialized OLTP or OLAP databases.

8. Kambayashi et al., (2023) proposed a novel concurrency control protocol named Shirakami, designed to handle both long and short transactions in real-world workloads efficiently. Shirakami has two sub-protocols: Shirakami-LTX for long transactions based on multi-version concurrency control, and Shirakami-OCC for short transactions based on Silo. It integrated these protocols using the write preservation method and epoch-based synchronization. Shirakami-LTX exploited a larger scheduling space (MVSR) than conflict serializability, reducing false positives and introducing order forwarding to handle conflicting long transactions without aborting. Shirakami-OCC is optimized for short transactions, retaining Silo's performance. The model reduced false positives, concurrent processing of long and short transactions and optimizations like read area declaration and on-demand version order determination. Yet, evaluating Shirakami on real-world benchmarks like BoM and telecommunication billing is not discussed in the research.

9. Nguyen et al., (2023) proposed a novel approach called Fair Thread ID (FairTID) for implementing transaction prioritization in real-time database systems. The proposed FairTID methods assigned thread IDs as timestamps to transactions, eliminating the need for a centralized counter. The model is implemented and evaluated on FairTID in a 2-phase locking protocol, maintaining the prioritization inherited from the Wound-Wait scheme. The

results show up to 1.5 times higher throughput, 1.6 times lower latency, and a 1.67 times lower deadline-miss ratio compared to the baseline protocol. However, the performance improvements were significant only when contention levels were not high.

10. Guo et al., (2024) proposed a new architecture for data sharing among heterogeneous blockchain using a cross-chain hub based on Trusted Execution Environment (TEE). The model introduces a hybrid concurrency control protocol to address performance variations among blockchain. It deploys a cross-chain hub on a TEE-enabled server to link diverse blockchain. A query-execute-commit mechanism ensures the isolation of data and processing logic. To enhance data security and confidentiality, the model implements key exchange and access control protocols. It proposes a hybrid concurrency control protocol that utilizes optimistic control for high-throughput blockchain and pessimistic control for low-throughput ones. The system achieves reasonable throughput and scalability in cross-chain data sharing, resulting in an execution time of 15798ms and conflict detection time of 1.225ms. Yet, the model performance may be limited by the slowest blockchain involved in data sharing.

2.2.4. Inferences from Literature Methods

Leveraging the strength and increasing performance in the concurrency control techniques requires different transactional characteristics and workload patterns along with a balance in allocating the resources, and maintaining the transactional request. The review inferred that the concurrency control in a distributed database environment combined with the optimization enhanced the performance and robustness of the system. The limitations

1. Integrating the Multiple concurrency control mechanism without increasing the complexity of the system is the primary challenge in hybrid concurrency control.
2. Coordinating various approaches in a highly concurrent environment increases the synchronization overhead and degrades the consistency.
3. The model adds to space complexity by storing multiple types of information, including transaction status and details, within the system.

The proposed research primarily focuses on enhancing the hybrid concurrency control methods within the DDB environment. The research progresses the hybrid concurrency control methods in DDB by integrating hybrid methods and utilizing nature-inspired search algorithms to effectively improve efficiency and mitigate challenges encountered in the literature.

2.2.5. Research gaps

•Need to improve concurrency control algorithms in database systems, particularly in centralized and distributed environments without degrading the performance of the system.

•Existing concurrency control methods exhibit limitations such as high abort rates, long execution times, and scalability issues.

•Lack of exploration of factors such as system load and dynamic environments in evaluating the execution of concurrency control techniques.

•Limited investigation of the system load and complexity on the performance and scalability of hybrid concurrency control techniques in distributed database systems.

•Lack of research in the optimization of concurrency control protocols for dynamic transaction processing environments, such as cloud databases, where workload fluctuations and resource constraints are common challenges.

2.3. DISTRIBUTED CONCURRENCY CONTROL ALGORITHMS

Distributed concurrency control algorithms encompass a range of methods and approaches utilized to handle concurrency in DDB systems. In distributed setups where data is dispersed across various nodes or sites, preserving the ability for transactions to operate simultaneously without causing conflicts and upholding the integrity and consistency of the data. These algorithms are designed to facilitate the coordination of access to shared data among transactions executing on disparate nodes within the distributed system.

1. Distributed Locking:

In distributed environments, the concept of distributed locking builds upon traditional locking mechanisms. It functions by acquiring and releasing locks on data items distributed across multiple nodes to regulate concurrent transaction access (Wang et al., 2017). The objective of distributed locking is to prevent

conflicting operations on the same data item, thereby ensuring that transactions can proceed without interference from one another.

2. Timestamp Ordering:

Timestamp ordering is a method where each transaction receives a unique Timestamp reflecting its initiation time. These timestamps are used to sequence transactions, determining their order of execution (Yu et al., 2016). This method ensures that transactions with conflicting operations are coordinated and executed in a manner that upholds consistency within the database.

3. Optimistic Concurrency Control (OCC):

OCC enables transactions to progress without immediately locking data items. Instead, it identifies conflicts during the commit phase by comparing the state of the database when the transaction started with its state at the time of committing. This approach is particularly effective in scenarios with minimal contention and infrequent conflicts, making it well-suited for such environments.

4. Multi-Version Concurrency Control (MVCC):

MVCC is a method applied in database systems to enable simultaneous access by multiple transactions. MVCC maintains multiple versions of these items without locking the data items. Every transaction accesses a stable snapshot of the database, ensuring isolation from other transactions (Ding et al., 2018). MVCC grants each transaction its version of data items, thus eliminating the necessity for locking, which reduces contention and enhances concurrency within the system.

5. Two-Phase Locking (2PL): 2PL, as an extension of the conventional phase locking protocol, guarantees serializability by enforcing transactions to follow a two-stage locking process. In the growing phase, transactions can acquire locks but are not permitted to unleash them and in the shrinking phase, transactions can unleash locks but are restricted from acquiring new ones. This mechanism ensures that transactions retain all essential locks until either they commit or abort, effectively preventing conflicts and upholding consistency throughout the process.

These algorithms help in managing concurrency in distributed database systems, allowing transactions to execute concurrently while ensuring data consistency and integrity across multiple nodes (pandey et al., 2018). Each algorithm has its advantages and disadvantages, and the decision of the algorithm relies on factors

such as the system's architecture, workload characteristics, and performance requirements.

2.4. STRUCTURE OF DISTRIBUTED TRANSACTIONS

The structure of distributed transactions denotes the organizational framework and components involved in maintaining and managing transactions across multiple nodes or sites in a distributed database environment. In DDBMS, synchronizing concurrent user transactions poses the challenge of adapting both serializability and concurrency control mechanisms to operate efficiently within a distributed framework. Global serializability necessitates that the execution of transactions at every site is seriable and the order in which the transactions are serialized across all sites remains constant. The DDBMS protocols address the limitation of restoring a failed site database to a consistent state. The protocols are designed to handle the complexities of distributed environments, ensuring that the system can recover and maintain data consistency across all the sites.

In DDBMS, each transaction originates from the master process (MP) which is located in the site of origin. This MP then establishes a group of cohort processes (COP) to perform the necessary processing tasks for executing the transaction. DDBS query processing strategies prioritize accessing data at its local site rather than a remote site. For each site where the transaction accesses data, there exists at least one corresponding cohort process. DDBS implementations vary in their approach to parallelism during query execution. When data is replicated across sites, each cohort responsible for updating any data item is associated with one or more update processes (UP) at other sites. In the distributed transactional system, each cohort is equipped with an update process localized to remote sites where data updates occur.

These update processes are utilized for maintaining concurrency control, facilitating communication among themselves, and sharing updates during the initial phase of the commit protocol. Upon completion of its designated query, a cohort signals its execution completion to the master (Wei et al., 2015). After, receiving receipt of such signals from all cohorts, the master initiates the commit protocol by dispatching “prepare to commit messages” to all relevant sites. Upon the decision to commit, a cohort notifies the master with a “prepare" message, prompting the master to relay commit directives to each cohort post-receipt of "prepared" messages from all cohorts. This protocol concludes with the master receiving "committed" confirmations from all cohorts. However, if any cohort

encounters an inability to commit, it returns a "cannot commit" message during the initial phase, prompting the master to transmit "abort" rather than "commit" directives during the subsequent phase. The presence of replica update processes renders the commit protocol a nested two-phase commit protocol. further enhancing the system's resilience and efficiency.

Two-Phase Commit Protocol (2PC):

2PC is applied to ensure atomicity in distributed transactions. It has two distinct phases namely the preparation phase and the commit phase. In the preparation phase, the coordinator prompts each participant to prepare for committing the transaction. After verifying readiness from all participants, the coordinator advances to the commit phase, directing each participant to execute the transactions commit (Gupta et al., 2018). If any participant encounters difficulty in preparing or committing, the coordinator initiates a rollback process to maintain transaction consistency, ensuring either a complete commit or abort across all participants.

Recovery Mechanisms:

These mechanisms are used to handle failures that may arise during the transaction processing. The mechanisms namely checkpoints, recovery managers and logging are used for maintaining a system. This method ensures the system's integrity by restoring the consistency state of the system and also ensures the durability of committed transactions.

2.5. MODELING OF DDBMS

The model of DDBMS consists of six components which help enhance the performance and ensure the properties of the data items (Grohmann et al., 2020).

1. **Source:** A source is employed to create and generate transactions concurrently tracks the performance information and also maintains transactional integrity at the site level.

2. **Transaction Manager:** this role involves receiving transactions from their source and patterning their execution. It also checks how transactions are carried out, including managing their behaviour throughout the execution process.

3. Concurrency Control Manager: It involves implementing concurrency control algorithms.

4. Resource Manager: This is responsible for representing physical resources such as CPUs, disks and files enabling operations such as writing to or accessing data or messages from these resources. This additionally represents the computational and I/O capabilities on each site. This structure additionally represents the computational and input/output (I/O) capabilities of each site.

5. Network Manager: The network manager contains the representation of the communication network model. It operates under the assumption of a LAN setup, where the transmission time of messages over the network is considered minimal or insignificant.

6. Sink: The sink manages the collection of statistics regarding completed transactions.

Figure 2.5 illustrates the DDBMS model structure and the processes are explained.

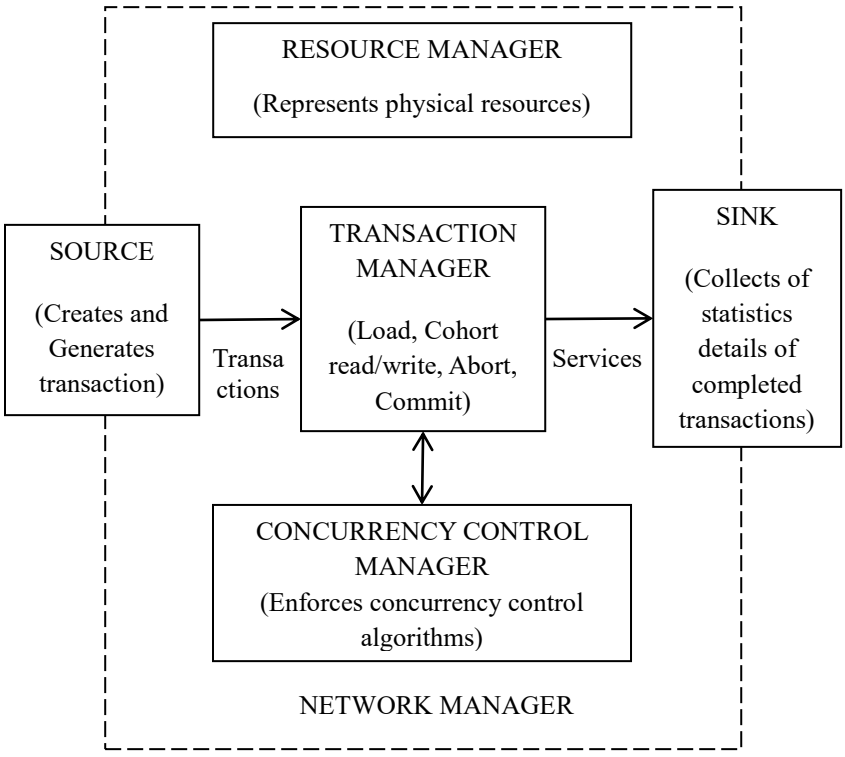


Figure 2.5 Modeling of DDBMS

2.6 .CONCURRENCY CONTROL PROTOCOL FOR THE DDBMS MODEL

Concurrency control protocol for DDBMS models encompasses a comprehensive set of rules and mechanisms to manage concurrency, ensure data consistency, and optimize performance in a distributed database environment. Figure 2.6 represents the types of Concurrency control protocol

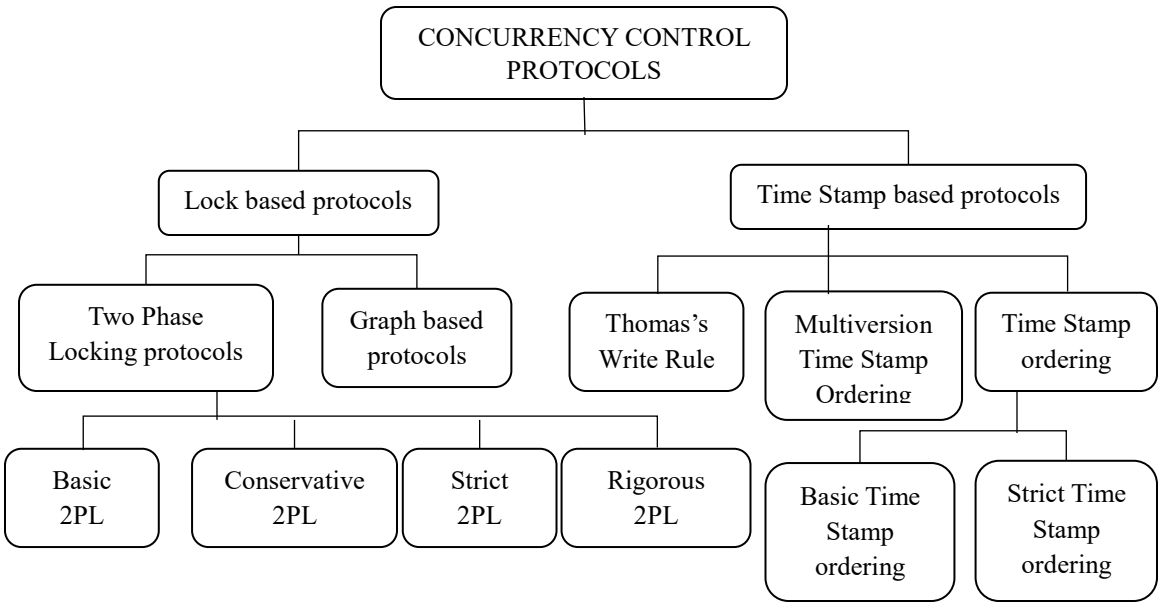


Figure 2.6 Concurrency control protocol

1. Lock based protocols

Lock-based protocols represent a category of strategies employed within database management systems to regulate simultaneous access to shared data by multiple transactions. Their primary aim is to guarantee that transactions obtain suitable locks on data elements, to prevent conflicts and uphold data consistency. The lock-based protocols are divided into two phase locking protocols which divided in to Basic 2PL, Conservative 2PL, Strict 2PL and Rigorous 2PL

1.1 two phase locking protocols: divided in to

A. Basic two-phase locking protocol (Basic 2PL):

A transaction is considered to be a two-phase locking protocol when all locking operations occur before any releasing operation. This protocol ensures serializability in a centralized database when transactions run concurrently. It operates under the assumption that a transaction can exist in only one of two phases. It contains two phases namely Growing Phase and Shrinking Phase.

In the Growing Phase, a transaction is only permitted to acquire locks and is blocked from releasing any lock (Pandey et al., 2018). Even if the transaction

completes its work with a locked data item, it is unable to release the lock during this phase. The growing phase concludes when the transaction reaches the lock point where the transaction has obtained all the locks it may potentially need.

Once the transaction reaches the lock point, it enters into the shrinking phase. In the Shrinking Phase, the transaction is limited to releasing locks and cannot acquire any new ones. The shrinking phase begins after the transaction releases its first lock beyond the lock point. Subsequently, the transactions continue releasing all locks it has acquired without the ability to acquire new ones.

Figure 2.7 represents the mechanism of the growing phase and shrinking phase in basic 2PL concurrency control protocols.

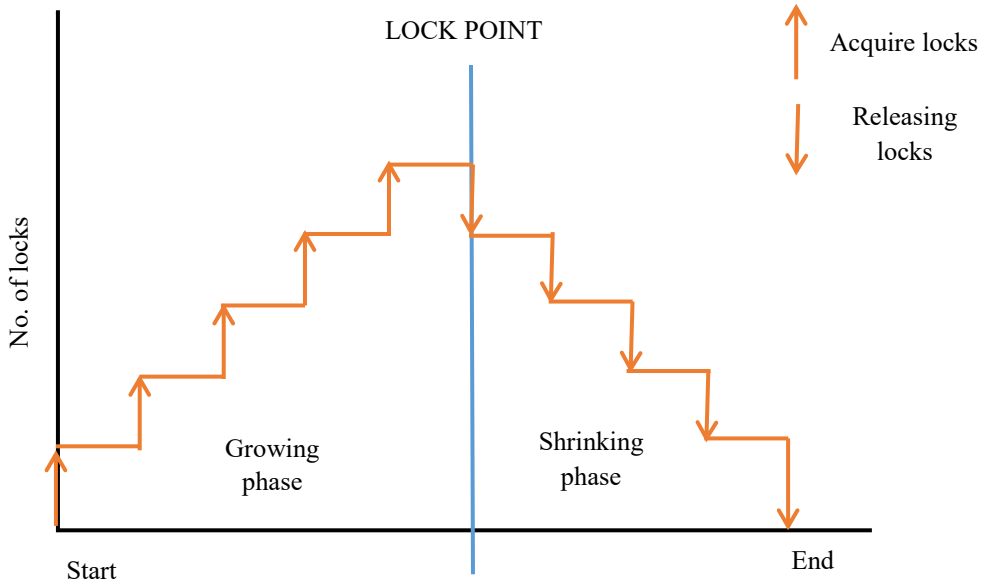


Figure 2.7 Basic 2PL locking protocol

B. Conservative two-phase locking protocol (C2PL):

C2PL requires transactions to declare all the locks they need upfront. This protocol acquires locks for all required items before a transaction begins execution by declaring the read and write sets are examined. If any of the elements in either the read or write set are already locked by other transactions, the transaction waits until those locks are released. While

this method is deadlock-free, it may not be feasible for a multi-user distributed database environment due to significant delays.

C. Strict two-phase locking protocol (S2PL):

In S2PL a transaction retains its write locks until it either aborts or commits, deadlock may still occur, but the protocol ensures recoverable schedules. Under strict scheduling, a transaction cannot access a data item that is locked until the last transaction that modified it either commits or aborts. Figure 2.8 represents the Strict two-phase locking protocol (S2PL)

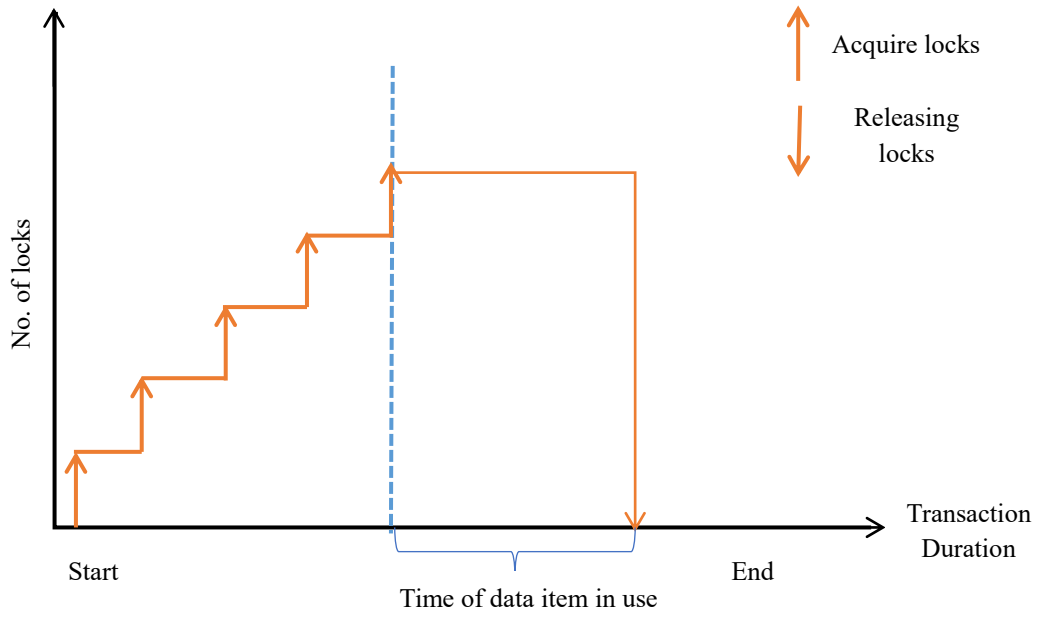


Figure 2.8 Strict two-phase locking protocols

D.Rigorous two-phase locking protocol (R2PL):

In the R2PL, locks are retained until a transaction either commits or aborts. A transaction remains in its expanding phase throughout its execution. Under this method, a transaction is prohibited from releasing any lock, until it commits. Other transactions may acquire shared locks on data items that the uncommitted transaction also holds shared locks, however, they cannot acquire any lock on data items.

1.2 Graph-based protocols:

Graph-based protocols are employed within DBMS to identify and prevent deadlocks between transactions. These protocols model the transactions and their interactions as a graph structure (Arora et al., 2016). In this representation, transactions are depicted as nodes, while conflicts between transactions are represented as edges within the graph. In this protocol, each transaction (T), can acquire an exclusive lock on data only once and must follow the rules that are defined below:

First Lock (R1): including the root node, T can initially acquire a lock on any data item.

Parent-Child Locking (R2): T can lock data (Q), only if it presently holds a lock on Q's parent.

Unlocking Flexibility (R3): Data items may be unlocked at any point during the transaction's execution.

Avoiding Re-Locking (R4): if data has previously been both locked and unlocked by T. It is prohibited from acquiring a lock on that data item.

2. Timestamp-based protocols:

Timestamp-based protocols are utilized to maintain and manage concurrent access to shared data by multiple transactions. The conflicts that arise while accessing the concurrent are rectified by using these Timestamp protocols.

2.1 Thomas's write rule:

This protocol is an extension of the timestamp ordering protocol and is used to solve conflicts between write and read operations. The rule defines that if a newer transaction has already recorded the value of an object, an older transaction need not execute its write operation as it will eventually be overwritten by the newer one. thomas's write rule, an enhancement of the basic timestamp ordering algorithm, deviates from enforcing conflict serializability but reduces the rejection of write operations by adjusting the criteria for the write_item(I) operation in the following methods:

R1: If the timestamp of the latest read operation on Data Item I, denoted as read_TS(I), is higher than the timestamp of transaction T (TS(T)), then transaction T is aborted and rolled back, rejecting the operation.
R2: If the timestamp of the latest write operation on data item I, denoted as write_TS(X), is higher than the timestamp of transaction T (TS(T)), then the write operation of transaction T is not executed. However, processing continues because another transaction with a higher timestamp than TS (T) has already written the value of X. Therefore, the write operation of T is considered out-dated.
R3: "If neither condition R1 nor condition R2 is met, the transaction T's write operation on data item I is executed, and the write timestamp of I (write_TS(I)) is assigned the timestamp of transaction T (TS(T)).

2.2 Multiversion timestamp ordering protocol:

Multiversion concurrency control represents an updated version of basic timestamp ordering techniques aimed at enhancing database performance in multiuser environments (Freitag et al., 2022). This algorithm preserves previous values of a data item whenever an update occurs, effectively maintaining a history of changes. In this approach, the system retains several versions (X1, X2, …,Xn) of each data item I. Whenever a write function is performed on I, a new version is generated. The data manager maintains X, which contains a record of its versions and a history log of the values assigned to X. During a read (I) operation, the scheduler not only decides when to initiate the read request to the data manager but also defines which version of I should be read. By using the write operation the new version of X is created. The Multiversion is governed by two rules
R1: When transaction T attempts to write to data item X, the system checks if there exists a version (i) of I with the high write timestamp (write_TS) that is lower than or equal to T's timestamp (TS(T)). If such a version exists and the read timestamp (read_TS) of version (i) is greater than T's timestamp, T is aborted and rolled back. Otherwise, a new version of I (xi) is created with its read timestamp set to T's timestamp, and its write timestamp set to TS (T).
R2: When transaction T tries to read data I, the system finds version i of X with the highest write timestamp that is lower than or equal to T's timestamp. It then returns the value of version i to T and updates the read

timestamp of version i to the maximum of T's timestamp and the current read timestamp of version. R2 ensures that a read operation will always be successful and never rejected.

2.3 Time Stamp Ordering divied in to :

A.Basic Timestamp ordering protocol:

In the Timestamp ordering protocol, a unique Timestamp (TS) which contains its start time is assigned to each transaction. The transactions are ordered based on their timestamp. The transaction with the higher Timestamp is considered to be the most recent or newer compared to the transactions with lower timestamp (Yao et al., 2015). Transactions are executed based on the timestamp, the transactions with the lower timestamp (older transactions) are given priority than the transactions with higher timestamp (newer transactions).

B. Strict time stamp ordering protocol:

The enhanced version of the basic timestamp ordering is called the strict time stamp ordering protocol which guarantees schedules that are both strict and serializable. It assures recoverability and conflict serializability. In this protocol, when a transaction (T), executes a read_item (I) or write_item (I) operation where Timestamp TS (T) > write_TS (I), the read or write operation is postponed until the transaction responsible for writing the value of I commits or aborts. The strict timestamp ordering protocol effectively circumvents deadlock issues.

2.7 . CHAPTER SUMMARY

This chapter provides a comprehensive introduction to DDBMS along with the concepts, characteristics and various types. It also delves into the DTP fundamentals and ACID properties essential for maintaining data integrity. This also explored concurrency control problems and strategies within DDBMS, including optimistic and pessimistic concurrency control methods. This reviewed recent and existing research works on DTP and concurrency control methods, emphasizing hybrid concurrency control approaches, while discussing their implications and limitations. The following chapters discussed the model and methodologies developed in this research to address and tackle the limitations and issues in the DDB environment and discusses the design of concurrency control mechanisms in DDBMS. It covers various distributed concurrency control algorithms like distributed locking, timestamp ordering, optimistic concurrency control, multi-version concurrency control, and two-phase locking protocols. It explains the structure of distributed transactions, including the roles of the master process, cohort processes, update processes, and the two-phase commit protocol for ensuring atomicity. Recovery mechanisms like checkpoints, recovery managers, and logging are also mentioned. The modeling of a DDBMS is described, consisting of components like sources, transaction managers, concurrency control managers, resource managers, network managers, and sinks. The chapter provides a detailed explanation of concurrency control protocols for DDBMS models, categorized into lock-based protocols, graph-based protocols, and timestamp-based protocols. The models involved in designing and implementing effective concurrency control in distributed database environments to ensure data consistency, integrity, and high

CHAPTER 3
HYBRID CONCURRENCY CONTROL TECHNIQUE FOR TRANSACTION PROCESSING IN DISTRIBUTED DATABASE SYSTEM

3.1 INTRODACTIN

A hybrid concurrency control method for transaction processing DDB systems is introduced, combining neural networks with a bio-inspired optimization algorithm. This method employs a Back Propagation Neural Network (BPNN) to manage lock-granting decisions and resolve conflicts by determining the winning transaction. To optimize the BPNN's performance, a metaheuristic optimization method known as the Bear Smell Search Algorithm (BSSA) is utilized to fine-tune its parameters. Historical data regarding transaction conflicts, resource contention, and system performance metrics are used to train the BPNN, enabling it to learn patterns and relationships between input parameters and concurrency control decisions. The strength of the BPNN lies in its ability to handle complex, non-linear relationships between input parameters and desired outputs. However, the effectiveness of the BPNN is dependent upon the quality of its training and the optimization of its parameters, such as weights and biases. By employing the BSSA, the proposed method aims to optimize the BPNN's parameters to enhance the accuracy and efficiency of concurrency control decisions.

3.2 HYBRID CONCURRENCY CONTROL MODEL FOR TRANSACTION PROCESSING

A Hybrid Concurrency Control Technique for Transaction Processing integrates both optimistic and pessimistic concurrency control methods to manage concurrent access to data in a database system. This approach aims to combine the benefits of both methods while mitigating their respective limitations.

The hybrid concurrency control technique combines elements of both optimistic and pessimistic methods to optimize performance and ensure data consistency. Transactions are initially executed optimistically, allowing them to proceed without acquiring locks. During the execution phase, the system monitors for potential conflicts using techniques such as validation checks or versioning mechanisms. If

conflicts are detected, the system switches to a pessimistic mode, where transactions acquire locks on data items to prevent further conflicts. Once conflicts are resolved and the transaction completes successfully, locks are released, and the system reverts to optimistic mode for subsequent transactions. Figure 3.1 demonstrates the overall Block diagram for hybrid concurrency control.

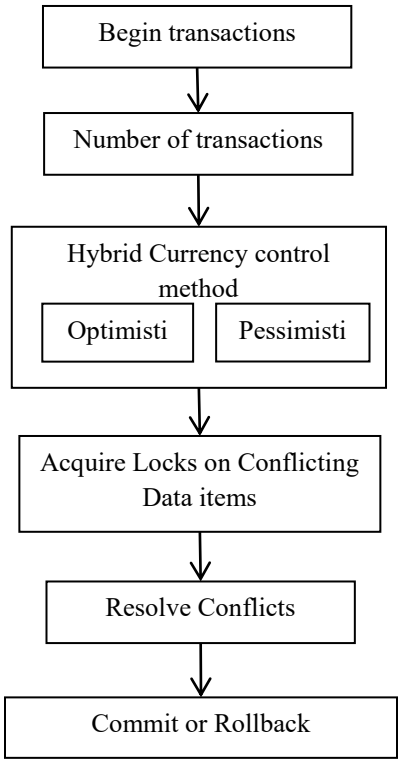


Figure 3.1 Block diagram for hybrid concurrency control

Transactions start optimistically, allowing for high concurrency and throughput. Validation checks are performed at the end of the transaction to detect conflicts. If conflicts are detected, the system switches to a pessimistic mode, acquiring locks to resolve conflicts and ensure consistency. Finally, transactions are committed or rolled back based on the outcome of conflict resolution. This hybrid approach combines the benefits of both optimistic and pessimistic concurrency control, providing a flexible and adaptive solution for transaction processing in database systems.

3.3 BACK PROPAGATION NEURAL NETWORK (BPNN)

The Backpropagation model serves as a crucial tool for training Artificial Neural Network (ANN) models, facilitating the learning process by adjusting weights to minimize the difference between predicted and actual outputs. This technique is integral to Multilayer feed-forward networks, commonly referred to as Backpropagation Neural Networks (BPNN). These networks comprise an input layer, a hidden layer, and an output layer, offering the capability to handle non-linear problems effectively (Li et al., 2017). The BPNN operates through two main phases: the forward pass and the backward pass. During the forward pass, data traverses the network layer by layer, undergoing computation using weights, biases, and activation functions. The resultant output is then evaluated against the expected labels to calculate the loss function. In the subsequent backward pass, the gradient of the loss function is figured utilising the chain rule of calculus, alongside weights and biases. These gradients are propagated backwards through the network, aiding in weight and bias adjustments to refine the model's performance. The forward propagation of the signal is denoted as,

$$S = (s_1, s_2, L, s_n)^T \tag{3.1}$$

Here, S is the signal transition from the input layer to the hidden layer

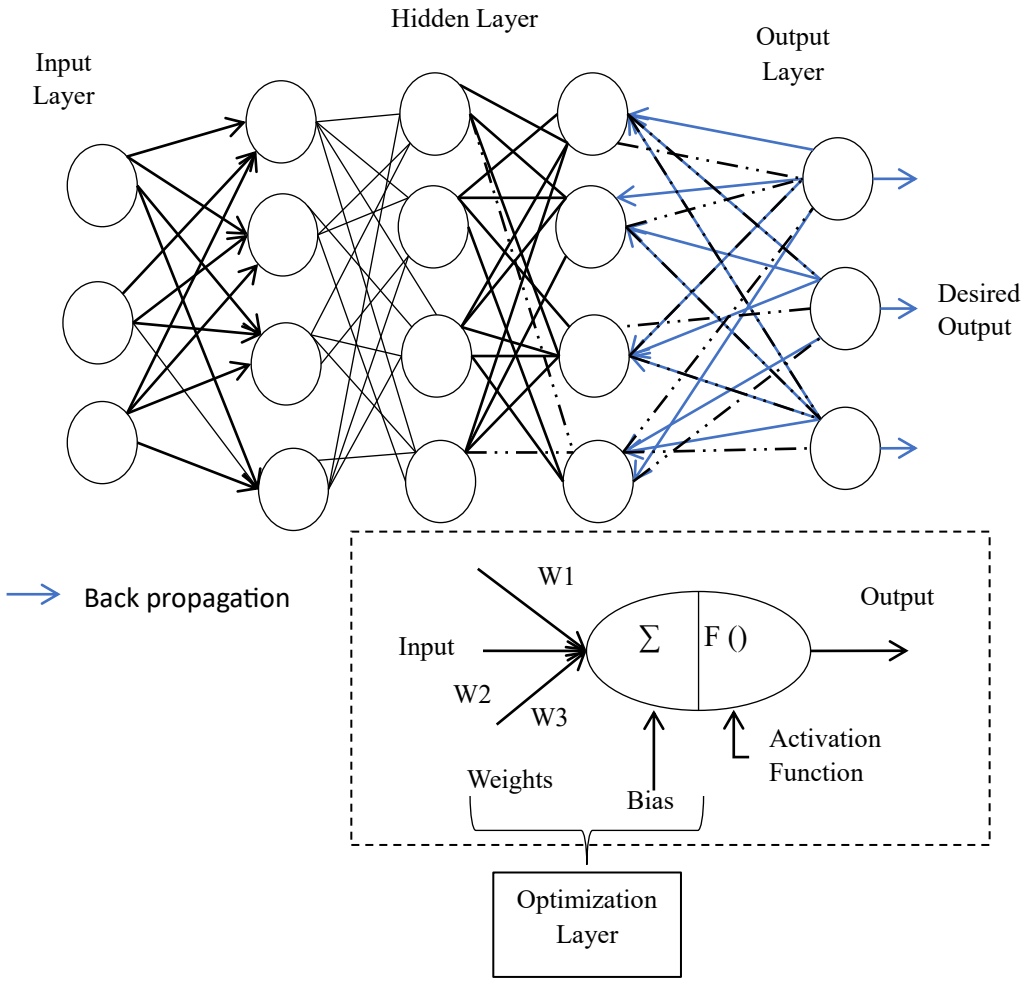


Figure 3.2 Architecture of the proposed model

By considering the neuron's activation function and threshold, the output of the neuron is determined to correspond to the layer structure. The output value hidden layer R_n for a neuron is mathematically expressed as,

$$R_n = a_1(\sum_{m=1}^{N} w_{m\,n} s_m - \theta_n) \quad (3.2)$$

Where $m\ and\ n$ refers to the $m^{th} and\ n^{th}$ neuron in the hidden layer, $w_{m\,n}$ refers to the weight connection between the $m^{th} and\ n^{th}$ neuron, θ_n refers to the threshold in the n^{th} neuron and a_1 refers to the activation function in the hidden layer.

The output v_l of the neuron in the output layer is denoted as,

$$v_l = a_2(\sum_{n=1}^{N} q_{l\,n} h_n - \theta_m) \quad (3.3)$$

Here, $q_{l\,n}$ represents the weight connection between the l^{th} layer and n^{th} neuron, θ_m refers to the threshold in the m^{th} neuron and a_2 refers to the activation function in the output layer.

The error function in the forward propagation is defined as,

$$E = \frac{1}{2}\sum_{l=1}^{L}(v_l - EO\,)^2 \tag{3.4}$$

Here, EO refers to the expected output. During the backpropagation to correct the acquisition error the threshold value is corrected based on the weight and bias, which is denoted as,

$$v_{l\,j} = v_{l\,j} + \Delta v_{l\,j}$$

$$\Delta v_{l\,j} = -\tau\frac{\partial E}{\partial v_{l\,j}} = -\tau\frac{\partial E}{\partial y_l} * \frac{\partial y_l}{\partial v_{l\,j}} = \tau(EO_l - y_l\,)y_l\,(1 - y_l\,)R_n \tag{3.5}$$

The error correction in the output layer Z_j is denoted as,

$$Z_j = (EO_l - y_l\,)y_l\,(1 - y_l\,) \tag{3.6}$$

The correction error e_n of each unit in the hidden layer can be denoted as

$$e_n = \left(\sum_{l=1}^{L} v_{l\,j} * Z_j\right)R_n\,(1 - R_n\,) \tag{3.7}$$

The learning process of the BPNN revolves around the iterative transmission of forward signals and the backpropagation of errors. The variation between the observed and the predicted results is adjusted by weight and bias along with the threshold values. To overcome the convergence and high volatility during the training process, which leads to the local optimum, the model uses the BSSA optimization algorithm. This algorithm is used to optimize and enhance the performance.

3.4 BEAR SMELL SEARCH ALGORITHM (BSSA)

The BSSA draws inspiration from the foraging behaviour of bears, leveraging their efficient strategies for locating and exploiting food sources. This algorithm mirrors the remarkable olfactory capabilities of bears, enabling them to detect and track smell over vast distances. By emulating the bear's ability to navigate toward promising smell trails while exploring new territories, the BSSA iteratively explores potential parameter configurations within a search space, guided by principles such as exploitation of promising regions and exploration of new areas. Assessing the quality of odors, particularly considering the interactions among odorant components, poses a challenge for traditional methods but aligns seamlessly with a bear's acute sense of smell (Ghasemi-Marzbali 2020). Bears possess an exceptionally large olfactory bulb, setting them apart from other species and amplifying their olfactory prowess. This enlarged olfactory bulb acts as a crucial receptor for detecting odors and transmitting sensory information through the olfactory tract to the brain.

This model aims to compute the glomerular activity relationship among odor components provided as inputs. Essentially, it generates similarity indices between received odors. It calculates similarity measures between the odors detected. Based on these similarity values, bears can determine their next movement direction. The BSSA relies on the bear's olfactory senses to perceive diverse scents in its surroundings. Each scent serves as a directional cue for movement, as every element within the environment emits a unique odor, guiding the bear's navigation. The distinct odor for the craved food is the final optimal solution. Let $O_i = [oc_i^1, oc_i^2 \ldots . oc_i^j \ldots oc_i^k]$ be the i^{th} odor received with k molecules/components. The bear receives n breathing odors, so the solution of breathing odors is initialized in a matrix $OM = [O_i]_{n \times k} = \left[oc_i^j\right]_{n \times k}$ format, which is given by,

$$OM = \begin{bmatrix} oc_1 \\ oc_2 \\ \vdots \\ oc_i \\ \vdots \\ oc_n \end{bmatrix}_{n \times k} = \begin{bmatrix} oc_1^1 & oc_1^2 & \ldots & oc_1^j & \cdots & oc_1^k \\ oc_2^1 & oc_2^2 & \cdots & oc_2^j & \cdots & oc_2^k \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ oc_i^1 & oc_i^2 & \cdots & oc_i^j & \cdots & oc_i^k \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ oc_n^1 & oc_n^2 & \cdots & oc_n^j & \cdots & oc_n^k \end{bmatrix}_{n \times k} \quad (3.8)$$

Here, O_i contains the components such as smell from trees, animals and various other environmental factors. The animal carcass smell is considered to be the target smell for the bears and oc_i^j is the j^{th} components of i^{th} odor.

In accordance with the breathing function and glomerular layer process in the sniff cycle, DS_i^j represents the j^{th} odor components in the i^{th} odor. This can be mathematically represented as

$$DS_i^j = \begin{cases} MG_i\,(T - T_{inhale}) + DS_i^{T_{inhale}}, & T_{inhale} \le T \le T_{exhale} \\ DS_i^{T_{exhale}} \exp\left(\frac{T_{exhale}-T}{\gamma_{exhale}}\right), & T_{exhale} \le T \end{cases} \quad (3.9)$$

Here, T_{inhale} , T_{exhale} are the inhalation time and exhalation time and γ_{exhale} represents the constant value for exhalation time. The complete breathing cycle remains constant, which either matches the length of variable k or the length of the i^{th}odor. Based on the timing of inhalation and exhalation, the odor components are separated into two different groups.

The receptor's sensitivities, identity and odor absorption are combined for the i^{th} mitral. When $DS_i^j = 0$, this indicates that there is no odor identified by the olfactory epithelium, before the next inhalation. The value of MG is calculated as,

$$MG_i(O_i) = \frac{1}{k}\sum_{j=1}^{k} f\left(oc_i^j\right), \quad f\left(oc_i^j\right) = \begin{cases} 1, & \vartheta \le oc_i^j \\ 0, & \vartheta > oc_i^j \end{cases} \quad (3.10)$$

Here, ϑ refers to the threshold value which is based on average odor information from sets and k refers to the odor length in i^{th}odor. The collected information is sent to the granular layers and mitral to find the optimal odor. These formulations replicate the neural dynamics observed in the granular and mitral layers.

$$\dot{X} = -H_0\,\varphi_y(Y) - \alpha_x X + \sum L_0\,\varphi_x(X) + DS \quad (3.11)$$

$$Y = W_0\varphi_x(X)^{\cdot} - \alpha_y Y + DS_c \quad (3.12)$$

Here, $X = \{x_1, x_2, \dots, x_n\}$, and $Y = y_1, y_2, \dots, y_n$ are the activities of granular and mitral cells, $DS = \{ds_1, ds_2, \dots, ds_n\}$ represents the external inputs to the mitral cells, $DS_c = \{ds_{c1}, ds_{c2}, \dots, ds_{cn}\}$ refers to the central input to granule cells, $\varphi_x(X) = \{f_x(x_1), f_x(x_2), \dots, f_x(x_n)\}$ refers outputs of the mitral cells and $\varphi_y(Y) = \{f_y(y_1),\ f_y(y_2), \dots, f_y(y_n)\}$ refers to the output of the granule cells,

$\alpha_x \; and \; \alpha_y$ are the constant values of mitral and granule cells and the constant value is set as 0.14, 0.29 and $f_x \; and \; f_y$ refers to the function which stimulates the output for the mitral and granule cells. The value $f_x(x) \; and \; f_y(y)$ are formulated with the constant value and formulated as

$$f_x(x) = \begin{cases} 0.14 + 0.14 \tanh\left(\frac{x-\vartheta}{0.14}\right), & x < \vartheta \\ 0.14 + 1.4 \tanh\left(\frac{x-\vartheta}{1.4}\right), & x \geq \vartheta \end{cases} \tag{3.13}$$

$$f_y(y) = \begin{cases} 0.29 + 0.29 \tanh\left(\frac{x-\vartheta}{0.29}\right), & x < \vartheta \\ 0.29 + 2.9 \tanh\left(\frac{x-\vartheta}{2.9}\right), & x \geq \vartheta \end{cases} \tag{3.14}$$

In the equation (3.13) and (3.14), the values of $H_0, L_0 \; and \; W_0$ refers relationship between the granular and mitral cells with the synaptic-strength connection matrixes. This is calculated as,

$$H_0{}_i^j = \begin{cases} \frac{rand()}{C_h}, & 0 < d_i^j < C_h \\ 0, & C_h < d_i^j \end{cases} \tag{3.15}$$

$$W_0{}_i^j = \begin{cases} \frac{rand()}{C_w}, & 0 < d_i^j < C_w \\ 0, & C_w < d_i^j \end{cases} \tag{3.16}$$

$$L_0{}_i^j = \begin{cases} \frac{rand()}{C_l}, & 0 < d_i^j < C_l \\ 0, & C_l < d_i^j \end{cases} \tag{3.17}$$

Where, $C_l, C_h \; and \; C_w$ are connection constants, $rand()$ refers to the random value ranges between [0, 1] and d_i^j indicates the space between i^{th} and g^{th} odors based on their data, where g^{th} odor is considered to be the desired smell (Global solution), which indicates the distance between the local solution (representing each odor) and the global solution (representing the target scent).

The optimization process integrates a mechanism by using the global solution to ensure and enhance the exploitation. When all the information is gathered from the activity and sent to the brain by neurons, the separation process starts based on the dissimilarity assessment. This dissimilarity process is done based on the Pearson correlation, which helps the bear to move to the next position based on the odor.

The factors used are probability odor components (POC), probability odor fitness (POF) and odor fitness (OF) which are formulated as,

$$POC = \frac{O_i}{\max(O_i)} \tag{3.18}$$

$$POF = \frac{OF_i}{\max(OF_i)} \tag{3.19}$$

The dissimilarity between two odors can be determined through calculations using expected odor fitness (EOF) and distance odor components (DOC) formulations as outlined below:

$$DOC_i = 1 - \frac{\sum_{i=1}^{k}(POC_j^1 - POC_j^2)}{\sqrt{\sum_{i=1}^{k}\left(POC_j^1 - POC_j^2\right)^2}} \tag{3.20}$$

$$EOF_i = |POF_i - POF^g| \tag{3.21}$$

The equations provided describe the potential movements within the algorithm. Essentially, these formulas elucidate the connection between odors and how they guide candidates toward the desired positions. The method demonstrates that the outputs of the brain determine an appropriate direction for the next position. Within each region of the grid, the distances between all odors are computed using two thresholds, denoted as $\aleph_1$ and $\aleph_2$. Consequently, the subsequent odors can be computed as follows:

$$O_{k+1} = \begin{cases} OD_{1,i} \times O_k - rand \times OD_{2,i} \times (O_k - O_{best}), & DOC_i \leq \aleph_1 \; and \; EOF_i \leq \aleph_2 \\ OD_{3,i} \times O_k - rand \times OD_{4,i} \times (O_k - O_{best}), & otherwise \end{cases} \tag{3.22}$$

$$OD_{1,i} = -EOF_i \frac{2 - DOC_i}{\aleph_1}, \quad OD_{2,i} = -EOF_i \frac{2 - DOC_i}{\aleph_2} \tag{3.23}$$

$$OD_{3,i} = -EOF_i \frac{2 - DOC_i}{\aleph_1}, \quad OD_{4,i} = -EOF_i \frac{2 - DOC_i}{\aleph_2} \tag{3.24}$$

ALGORITHM FOR BSSA

1. The parameters are initialized: no. of population, maximum iterations ($iter_{max}$)
2. The population is generated with dimension $OM = N \times D$ in search space.
3. The fitness value for each row matrix (OM) is evaluated.
4. The current global solution O^g from the best odor is saved.
5. For iteration = $1 : iter_{max}$
    a. The maximum output for the glomerular activity (MG) is calculated by using Eq. (3.10) & (3.11)
    b. The DS value based on breathing function is attained by using Eq. (3.9)
    c. The values of $f_x(x)\ and\ f_y(y)$ are calculated as the cell function output using Eq. (3.13) and (3.14).
    d. The values of $H_0, L_0\ and\ W_0$ are calculated using Eq. (3.15)-(3.17) and solve Eq. (3.11)-(3.12)
    e. The vectors POC, POF and OF are defined using the Eq. (3.11)-(3.12)
    f. The vectors DOC and EOF are calculated using Eq. (3.18)-(3.19)
    g. The thresholds to calculate the distance between the odors are set as $\aleph_1$ and $\aleph_2$
    h. To calculate the distance, the coefficient $C_1\ to\ C_4$ values are calculated,
    i. For $i = 1: n$
        i. If $DOC_i \leq \aleph_1\ and\ EOF_i \leq \aleph_2$ then
            1. Generate the population using Eq. (3.23)
        ii. Else
            1. Generate the population using Eq. (3.24)
        iii. End

j. End for

6. Print the global outputs
7. End for

Figure 3.3 Graphical view of the olfactory system, since the direction of odors toward bear is important; therefore, only these types of odors are drawn but in real-world these odors released in all directions (Ghasemi-Marzbali 2020).

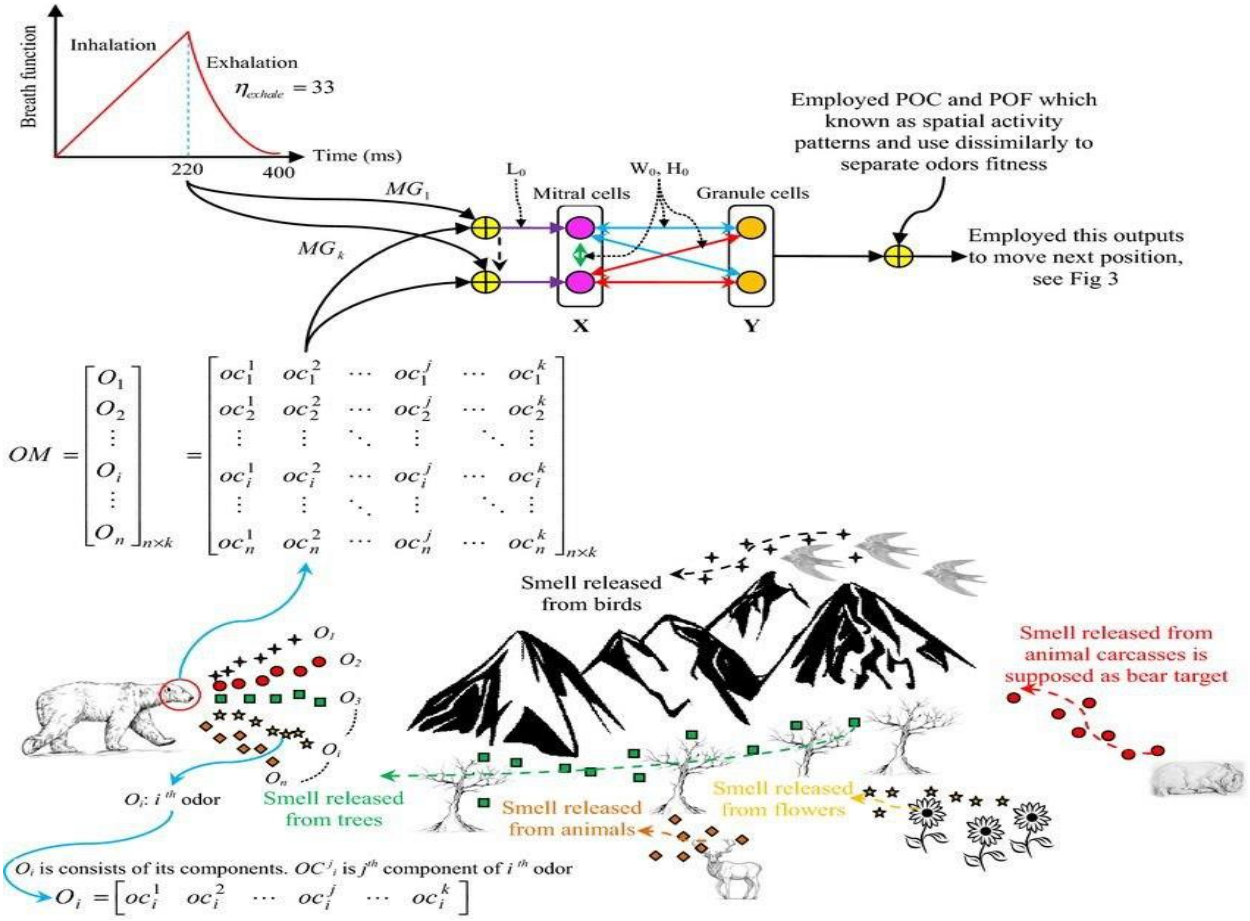


Figure 3.3 Graphical view of the olfactory system

3.5. BPNN-BAAS-BASED HYBRID CONCURRENCY CONTROL MODEL

The proposed BPNN-BSSN aims to increase accuracy and efficiency. The BPNN is used to lock and manage the lock decisions, along with resolving conflicts. This also helps in recognizing successful transactions. The parameters of BPNN are fine-tuned by utilizing BSSA. The BSSA is used to optimize transaction conflicts, resource contention, and system performance metrics based on the transaction history. The advantage of BPNN includes handling intricate and non-linear relations between inputs and output. By fine-tuning the weight and bias parameters the efficiency and the performance of the model is increased. The BSSA is enlisted to optimize these parameters, aiming to discover the optimal set that enhances the accuracy and efficacy of concurrency control decisions. Figure 3.4 demonstrates the flowchart for the proposed BPNN-BSSA model.

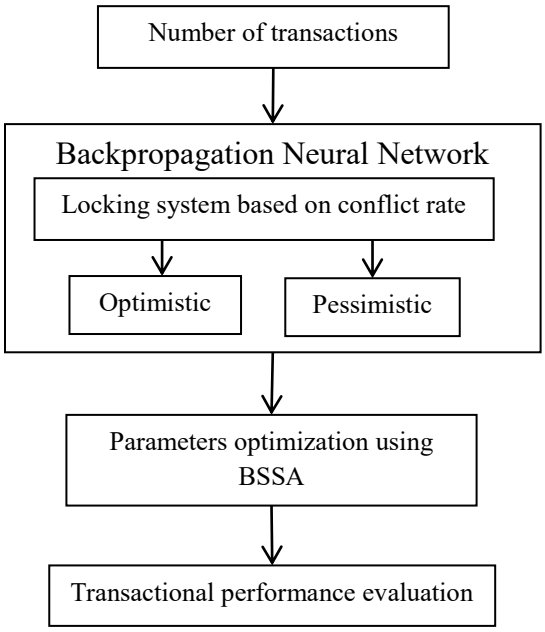


Figure 3.4 Block diagram for the proposed BPNN-BSSA model

For each transaction in the database, a distinct identifier is assigned to ensure the effectiveness and manageability of concurrency control techniques within the database. In this model, the database designates the transaction_id as the unique identifier. Each object within the database operates in two versions: the committed version, indicating successful execution and commitment, referred to as the old version, and the uncommitted version, denoting on-going execution, known as the new version. If objects from a transaction possess only one version and remain

unchanged for an extended period, they are considered inactive and are kept within hidden transactions. These hidden transactions persist until the transaction's writing process is finalized and committed.

The database system maintains records of all objects, including both committed and uncommitted transactions, as well as on-going transactions that have interacted with these objects through reads or writes without committing. Consequently, read or write locks are enforced on these objects. Whenever a transaction executes a write operation on an object, a new version of that object is generated. This new version persists until the transaction is committed. Upon completion of the transaction and satisfaction of all conditions, the previous version of the object is replaced. To manage conflict resolution, transactions maintain two lists, referred to as list1 and list2, which serve the purposes of sequential execution and deadlock detection, respectively.

When a conflict of type read-write (rw) occurs, where one transaction reads an object that another transaction writes a new version of, the following actions are taken:

The conflicting transaction Transaction1 is added to *list1* of Transaction2 with Type = R.

Transaction2 is added to *list2* of Transaction1 with Type = R.

In the case of a write-read (wr) conflict:

Transaction1 is added to *list1* of Transaction2 with Type = W.

Transaction2 is added to *list2* of Transaction1 with Type = W.

The commitment of Transaction 2 is postponed until the transactions in *list 1* of Transaction 2 are completed. Transactions in *list2* of Transaction1, with types W/R, simply await the termination of Transaction1. However, transactions of type-write in the *list2* of Transaction1 are dependent on the result of Transaction1. If Transaction 1 aborts, these transactions should also be aborted. The proposed algorithm considers transaction states as Ready, Waiting, and Commit.

The optimistic method outperforms the pessimistic method when conflict occurrences are infrequent. Determining the appropriate method for different scenarios requires calculating the conflict rate, which is influenced by the time frame within the database. A proposed algorithm dynamically selects between pessimistic and optimistic concurrency control approaches based on observed

conflict rates over time. Monitoring the number of transactions and conflicts within a specific timeframe is crucial. Two buffers are employed: one records transaction inputs, while the other logs transaction commands between conflicts. This systematic approach facilitates the tracking of transaction and conflict counts in each cell.

To determine and adopt optimistic and pessimistic the proposed model relies on conflict rate parameters defined as

$$E = \frac{N_c}{N_i} \tag{3.26}$$

Here, E is the conflict rate parameter, N_c is the number of conflicts in a certain timeframe and N_i number of input transactions in a certain timeframe.

If decision-making via the RS/WS is not feasible, the determination relies on the conflict rate and it is compared to the threshold value. After the committed transaction from the BPPNN model, the model uses the conflict rate value to determine the approach between optimistic and pessimistic. To determine the locking system the threshold value is set based on the conflict, in this stimulation the threshold value is set as 0.8. If the conflict rate is less than the threshold value ($E < 0.8$) then the optimistic approach is selected, whereas if the conflict rate is greater than a threshold value ($E > 0.8$) then the pessimistic approach is determined.

By combining the backpropagation algorithm with the BSSA, the training process of the neural network can potentially be optimized more effectively, by using the parameters Throughput (TP), Conflict rate (CR), Execution time (ET) and CPU time (CT).

$$FF = \alpha \times TP + \beta \times CR + \gamma \times ET + \delta \times CT \tag{3.27}$$

Here, $\alpha, \beta, \gamma \; and \; \delta$ are the random weight vectors along with the parameters Throughput (TP), Conflict rate (CR), Execution time (ET) and CPU time (CT).

By optimizing the parameters the model leads to better performance and faster convergence. The BSSA helps explore the search space more efficiently and find the optimal combination of weights and biases that minimize the error function. Figure 3.5 illustrates the flowchart of the hybrid concurrency control model.

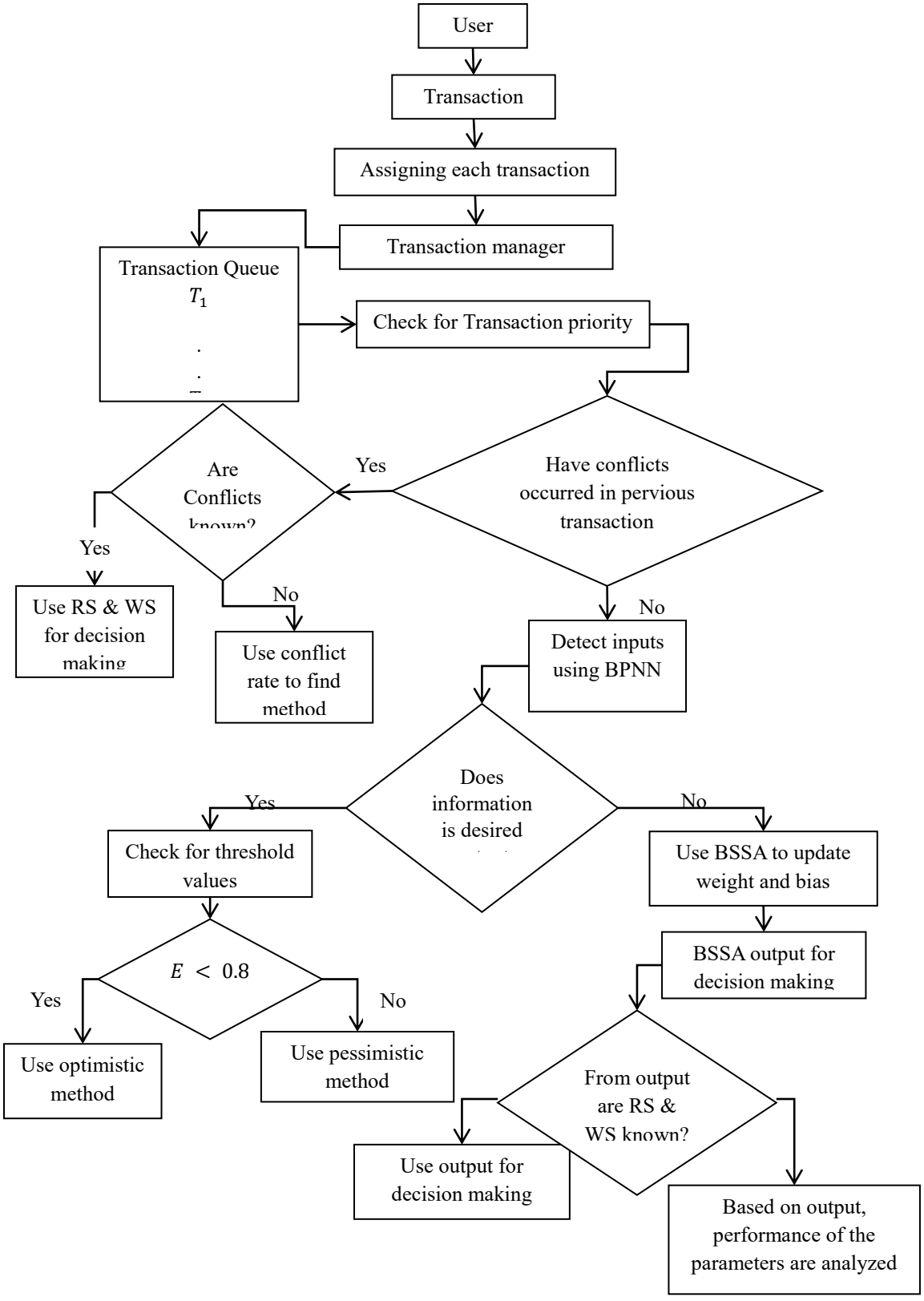


Figure 3.5 Flowchart of the hybrid concurrency control model

Algorithm for proposed model BPNN-BSSA

1. Start
2. Initialize parameters: Number of populations, maximum iterations
3. Generate population in the search space
4. Evaluate the fitness value for each row matrix
5. Save current global solution
6. For each iteration: Calculate maximum output for glomerular activity (MG)
7. Calculate DS value based on breathing function
8. Calculate values of activation functions $f_x(x)\ and\ f_y(y)$
9. Calculate $H_0, L_0\ and\ W_0$
10. Calculate vectors $POC, POF, and\ OF$
11. Calculate vectors $DOC\ and\ EOF$
12. Set thresholds for distance calculations
13. Calculate coefficients (C1 to C4)
14. Generate populations based on odor dissimilarity
15. Update global outputs

    Optimistic vs. Pessimistic Approach Determination:

16. Calculate conflict rate parameter (E) using Equation (3.26)
17. If decision-making using RS/WS is not possible, base the decision on the conflict rate
    a. Determine a locking system based on conflict rate and threshold value
    b. Compare E with a threshold value (0.8)
    c. If $E < 0.8$:
        i. Select optimistic approach
    d. Else $E > 0.8$:

i. Select pessimistic approach

e. End if

Concurrency Control Model and Transaction Processing:

f. Set a unique key for each transaction (transaction_id)

g. Maintain versions of objects: committed (old version) and uncommitted (new version)

h. Handle read-write (rw) conflicts and write-read (wr) conflicts

i. Manage transaction states: Ready, Waiting, Commit

18. End if

19. Parameters are fine-tuned

20. Performance is analysed

21. End for

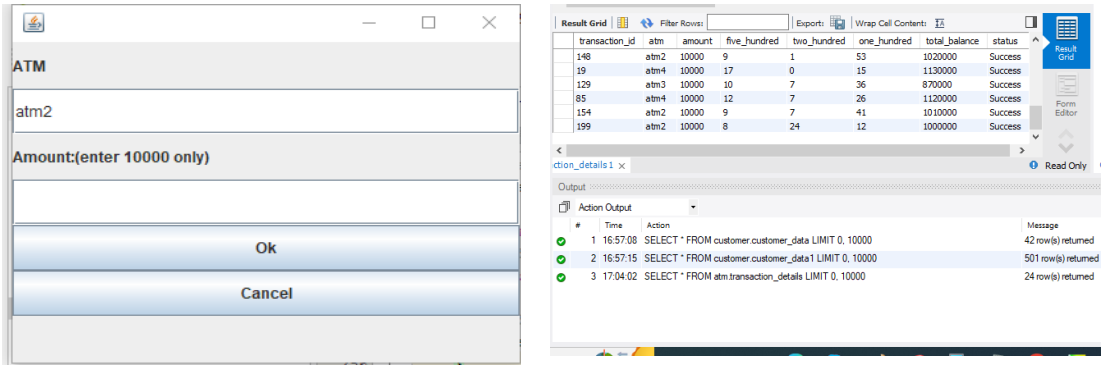
22. Stop

The proposed hybrid concurrency control model exploits the benefit of the BPNN and BSSA to ensure and enhance the efficiency, accuracy and performance of the database concurrency control model. The BPNN-BSSA model is employed for lock management, conflict resolution and transaction recognition along with the parameters optimization with the BSSA optimization.

3.6.IMPLEMENTATION

The banking simulation system comprises three interfaces: the ATM (Automate Tellar Machine)form, the Customer form, and the Employee form. These forms are designed to replicate real ATM processes. Integration with a database facilitates transaction management, while CSV file handling manages transaction data input. The banking system follows an object-oriented structure, featuring diverse functionalities. An ATM number is randomly generated to populate the corresponding ATM field. A constraint ensures withdrawal amounts are limited to 10,000; which is the permissible amount for stimulation. If any withdrawal amount exceeds the limit, the system triggers a warning message. The code validates withdrawal amounts, prompting the warning if necessary. Action listeners for update and cancel operations manage ATM transactions, involving retrieval of the

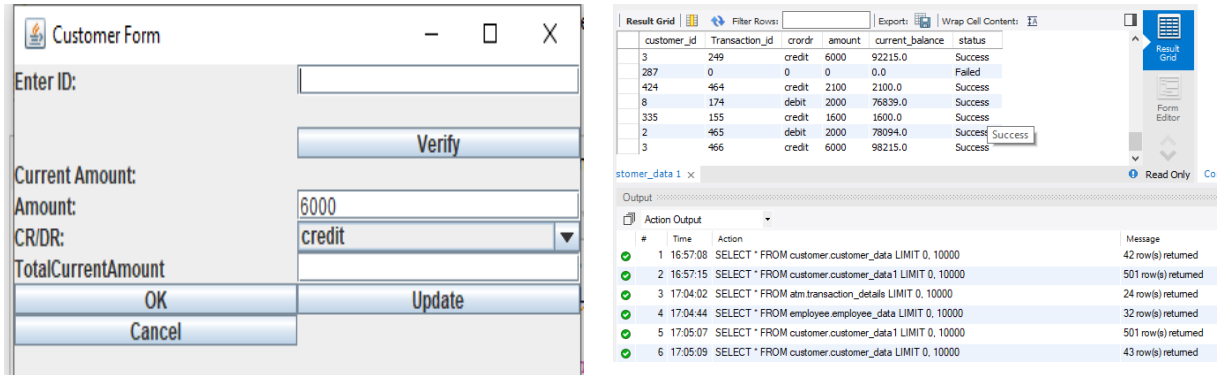
ATM number and withdrawal amount. Transaction details, including withdrawn amounts, total balances, and transaction status containing the success or failure, are updated in the database.



(a) ATM Form (b) ATM database

Figure 3.6 ATM interface

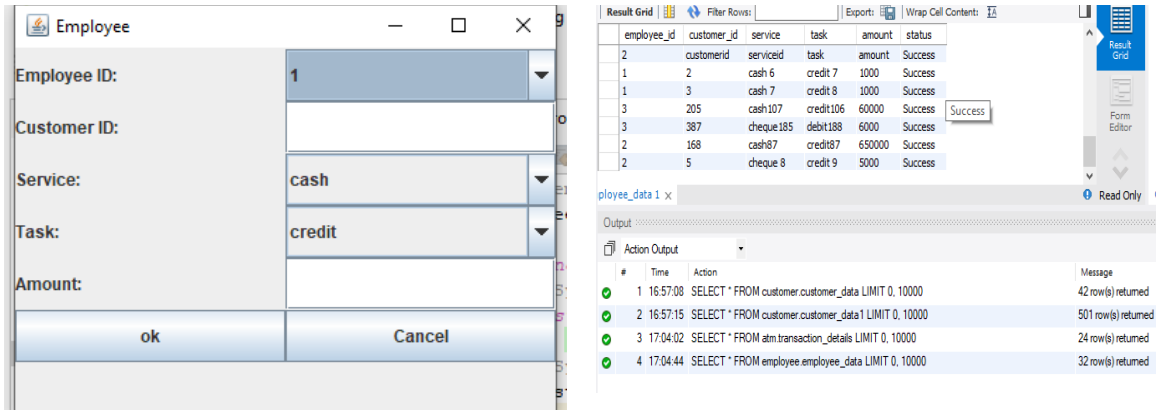
The customer form integrates design and operational elements for both database and user interface tasks. This model ensures to creation of an interface optimized for user interaction with transaction and account details. Customer form gathers and retains customer transaction information such as credit or debit activities, current balance, and transaction status. Upon connection to the database, it authenticates the customer ID and retrieves the current balance. Subsequently, the balance is computed based on the selected credit or debit option and the entered amount. In cases where the resulting balance would be negative, it alerts the user about insufficient balance. Transaction updates are then executed within the database if the translation update fails; this promotes the display of an error message. Database contents include customer ID, transaction ID, credit or debit details, transaction amount, and current balance. This represents the structure and functionality of the database and user interface, encompassing customer transaction management, validation procedures, database interactions, and error resolution. Additionally, it creates a user-friendly environment conducive to diverse transaction-related operations. Figure 3.7 represents the Customer Interface which includes the customer form and customer database.



(a) Customer Form (b) Customer database

Figure 3.7 Customer Interface

The employee application's functionalities and methods, encompassing database operations and user interactions, are outlined within the employee form or interface. These methods serve to establish connections, execute queries, and manage the creation and updating of employee details, including comprehensive exception handling. The application incorporates functionality to manage aborted transactions and retrieve relevant data. Within the employee form, essential fields such as employee ID, customer ID, services, tasks, amounts, and transaction details are present, covering outcomes like success, failure, and aborted transactions. All acquired details are securely stored within the database for future reference and processing. Figure 3.8 represents the Employee Interface which includes Employee form and Employee database.



(a) Employee form (b) Employee Database

Figure 3.8 Employee interface

3.7. CHAPTER SUMMARY

The chapter discussed a novel hybrid concurrency control technique for transaction processing in distributed database systems. It combines a BPNN with a bio-inspired optimization algorithm BSSA. The BPNN is used to manage lock-granting decisions, resolve conflicts and recognize successful transactions. The BSSA is employed to optimize the BPNN's parameters like weights and biases to enhance its accuracy and efficiency. The hybrid technique aims to combine the benefits of optimistic and pessimistic concurrency control methods. It uses a conflict rate parameter to dynamically select between the two approaches based on observed conflict rates over time. The proposed model maintains multiple versions of database objects, handles read-write and write-read conflicts through transaction lists, and manages transaction states like Ready, Waiting, and Commit. An implementation is described as involving an ATM simulation system with interfaces for ATM, customer, and employee transactions integrated with a database for transaction management.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 EXPERIMENTAL SETUP AND EVALUATION METRICS

The experiments were carried out using Netbeans IDE (version 17), SQL workbench 8.0 and JDK 20 on an Intel Core i5 processor with a CPU frequency of 1.8GHz, 8GB RAM, and the Windows 10 operating system. There are a total of 500 customer details which are stored in the database along with the ATM details and the Employee details are stored in the SQL workbench database. Once the transaction begins the other information including the type of transaction and the success or failure of the transaction is also stored in the workbench database.

To evaluate the performance based on the reliability and efficiency of transaction processing systems for various numbers of transactions the evaluation metrics are utilized. Metrics like MSE, Throughput (Mbps), Conflict Ratio, Failure Ratio, Execution Time (s) and CPU Utilization are used for performance evaluation.

MSE (Mean Squared Error) is a measure of the average squared difference between actual and predicted values.

$$MSE = \frac{1}{N} \sum_{I=1}^{N} (\hat{y}_i - y_i)^2 \qquad (4.1)$$

Here, N is the number of samples, y_i is the actual value and $\hat{y}_i$ is the predicted value.

Throughput measures the rate at which data is successfully transmitted over a network in megabits per second (Mbps).

$$Throughput = \frac{Total\ Data\ Transmitted}{Time\ Taken} \qquad (4.2)$$

Conflict ratio represents the ratio of conflicts to total transactions.

$$Conflict\ Ratio = \frac{Number\ of\ Conflicts}{Total\ Number\ of\ Transactions} \qquad (4.3)$$

The failure ratio indicates the ratio of failed transactions to total transactions.

$$Failure\ Ratio = \frac{Number\ of\ Failed\ Transactions}{Total\ Number\ of\ Transactions} \qquad (4.4)$$

Execution time measures the time taken to complete a task or process in seconds.

$$Execution\ time = IC \times CPI \times T \quad (4.5)$$

Here, IC – Instruction count, CPI – clock period instruction and T – Clock period

CPU utilization represents the percentage of time the CPU spends executing instructions.

$$CPU\ Utilization = \frac{Total\ CPU\ Time\ Used}{Total\ Elapsed\ Time} \times 100\% \quad (4.6)$$

4.2 PERFORMANCE RESULTS

The performances are evaluated under 25, 50, 75 and 100 Transactions. The MSE values fluctuate across different transaction levels. The values of throughput increase with the increase in number of transactions. The conflict ratio and failure ratio are relatively stable in different transaction levels, this indicates the system performance is consistent in terms of transaction failure handling and conflict resolution. The CPU utilization indicates that the system used consistent resource usage in various transaction levels. The execution time remains within the closed range with some variability. The performance metrics show that the system maintains stable performance across different transaction levels, with throughput and execution time being the most directly influenced by transaction volume.

Table 4.1 Performance metrics for a system under various numbers of transactions.

No. of transaction	MSE	Throughput (Mbps)	Conflict Ratio	Failure Ratio	Execution Time (s)	CPU Utilization
25	0.0000029707	3.355	0.4	0.04	88.402	11.67
50	0.0000002948	3.6909	0.44	0.06	99.653	11.71
75	0.0000052466	3.355	0.4	0.054	86.453	11.572
100	0.0000027564	3.523	0.42	0.05	97.475	11.56

Table 4.2 Comparative analysis of performance metrics with existing methods

Models/ Methods	MSE	Throughput (Mbps)	Conflict Ratio	Failure Ratio
HCC-MGSA-ART	0.00000376	2.174	0.98	0.71
HBOCC	0.00000369	1.365	1.67	0.4
SAOL –LRCD	0.00000389	2.479	1.20	0.45
RDMA-RCC	0.00000345	1.437	1.47	0.5
TCC –TDG	0.00000387	1.987	1.04	0.3
ES2PL	0.00000356	2.637	1.12	0.17
MCSI-NVM	0.00000373	1.40	1.11	0.2
DPC2-CD	0.00000312	3.026	1.30	0.7
RDMA-2PL	0.00000319	1.759	1.23	0.6
RDMA	0.00000343	2.6126	1.21	0.9
TEE –BCCH	0.00000367	1.427	1.29	0.1
Proposed model BPNN-BSSA	**0.00000275**	**3.523**	**0.42**	**0.05**

Table 4.2 provides a comparative analysis of performance metrics among various existing models with the proposed model BPNN-BSSA. The proposed model significantly outperforms all other models. In terms of MSE, the values decrease from the range 0.0000949436 to 0.0000331564, throughput increases with the range from 3.158 to 0.114, Conflict ratio decreases from range 0.62 to 3.158 and failure ratio decreases with the range from 0.15 to 0.85. The proposed model BPNN-BSSA demonstrates a superior performance in terms of accuracy, Lower conflict and failure rate along higher throughput, when compared to other models. This indicates that the model has higher accuracy for prediction, better synchronization, coordination and reliability. The overall performance of the model indicates its effectiveness.

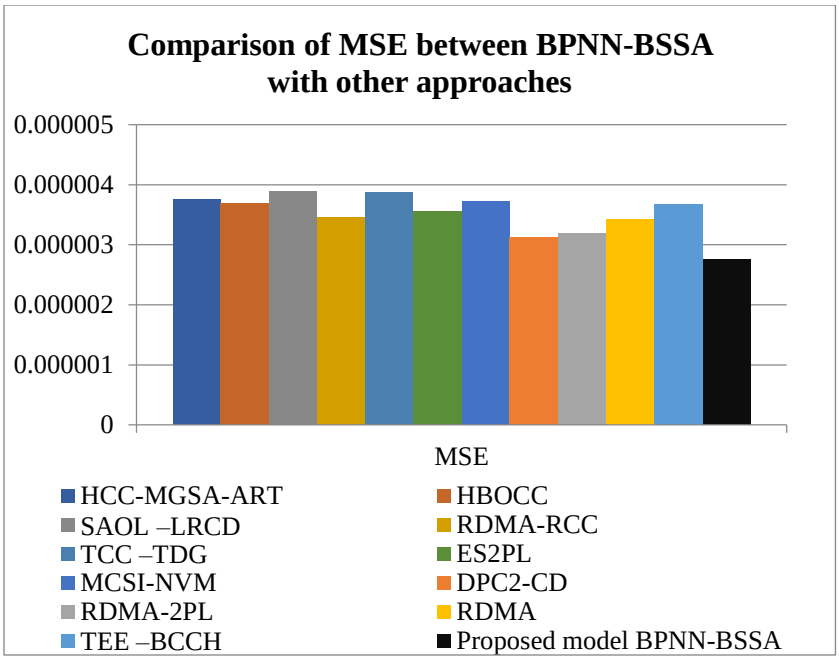


Figure 4.1 Comparison of MSE between BPNN-BSSA with other approaches

Figure 4.1 illustrates the comparative analysis between the proposed BPNN-BSSA and existing approaches. The BPNN-BSSA model has a lower MSE value when compared to other models. The MSE value of proposed BPNN-BSSA is 0.00000275, and it is 0.00000101, 0.00000094, 0.00000114, 0.0000007, 0.00000112, 0.00000081, 0.00000098, 0.00000037, 0.00000044, 0.00000068, 0.00000092, higher than HCC-MGSA-ART, HBOCC, SAOL –LRCD, RDMA-RCC, TCC –TDG, ES2PL, MCSI-NVM, DPC2-CD, RDMA-2PL, RDMA and TEE –BCCH methods.

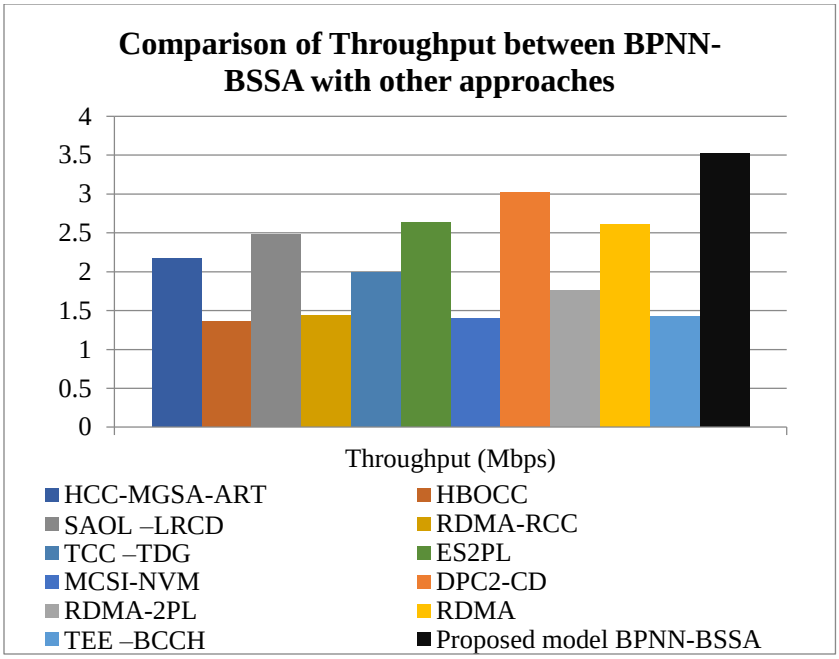


Figure 4.2 Comparison of Throughput between BPNN-BSSA with other approaches

Figure 4.2 illustrates the comparative analysis of throughput between the proposed BPNN-BSSA and existing approaches. The BPNN-BSSA model has a higher throughput value when compared to other models. The throughput values of the proposed BPNN-BSSA is 3.523 Mbps, and it is 1.349mbps, 2.158mbps, 1.044 Mbps, 2.086 Mbps, 1.536 Mbps, 0.886 Mbps, 2.123 Mbps, 0.497 Mbps, 1.764 Mbps, 0.9114 Mbps, 2.096 Mbps, lower than HCC-MGSA-ART, HBOCC, SAOL –LRCD, RDMA-RCC, TCC –TDG, ES2PL, MCSI-NVM, DPC2-CD, RDMA-2PL, RDMA and TEE –BCCH methods.

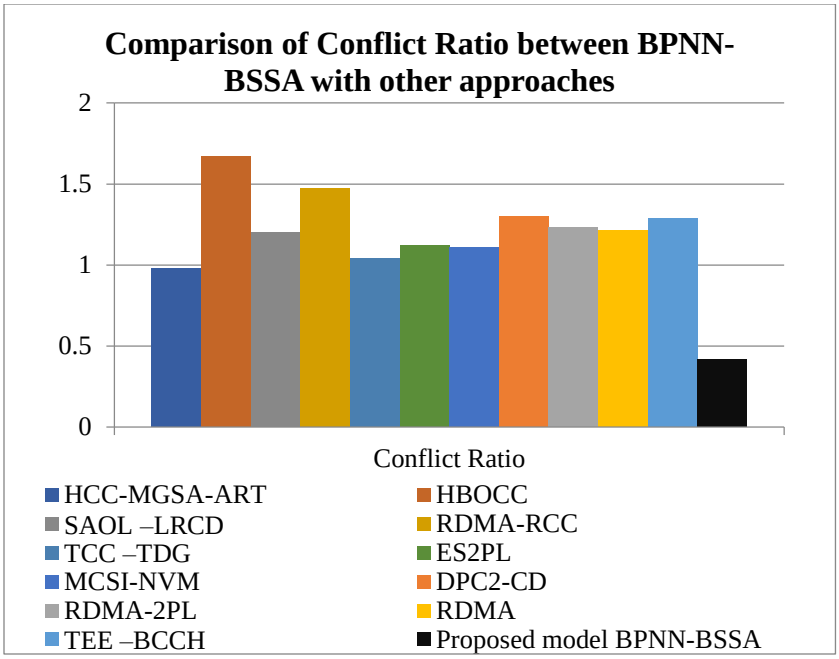


Figure 4.3 Comparison of Conflict Ratio between BPNN-BSSA with other approaches

Figure 4.3 illustrates the comparative analysis of the conflict ratio between the proposed BPNN-BSSA and existing approaches. The BPNN-BSSA model has a lower conflict ratio when compared to other models. The conflict ratio of the proposed BPNN-BSSA is 0.42, and it is 0.56, 1.25, 0.78, 1.05, 0.62, 0.70, 0.69, 0.88, 0.81, 0.79, 0.87, higher than HCC-MGSA-ART, HBOCC, SAOL –LRCD, RDMA-RCC, TCC –TDG, ES2PL, MCSI-NVM, DPC2-CD, RDMA-2PL, RDMA and TEE –BCCH methods.

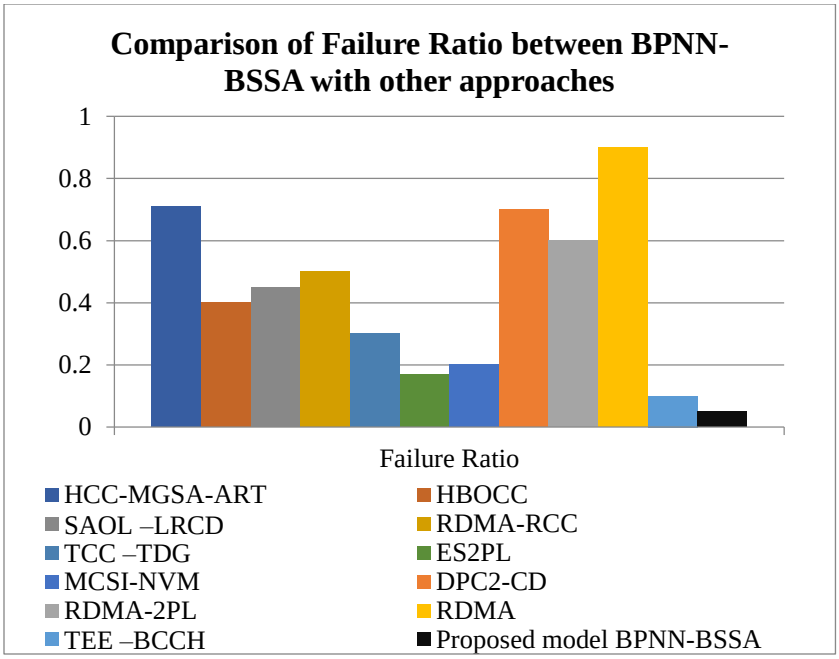


Figure 4.4 Comparison of Failure Ratio between BPNN-BSSA with other approaches

Figure 4.4 illustrates the comparative analysis of the failure ratio between the proposed BPNN-BSSA and existing approaches. The BPNN-BSSA model has a lower failure ratio when compared to other models. The failure ratio of proposed BPNN-BSSA is 0.05, and it is 0.66, 0.35, 0.4, 0.45, 0.25, 0.12, 0.15, 0.65, 0.55, 0.85, 0.05 higher than HCC-MGSA-ART, HBOCC, SAOL –LRCD, RDMA-RCC, TCC –TDG, ES2PL, MCSI-NVM, DPC2-CD, RDMA-2PL, RDMA and TEE – BCCH methods.

The BPNN-BSSA model exhibits superior performance in terms of MSE, with significantly lower values compared to other models. It also demonstrates higher throughput for faster data processing capabilities compared to other models. In terms of conflict ratio and failure ratio, the BPNN-BSSA model surpasses other models by maintaining lower values, indicating better synchronization, coordination, and system reliability.

4.3 DISCUSSION OF RESULTS

The hybrid concurrency control model integrates the advantages of the pessimistic and optimistic approach for transaction processing in a distributed environment. The model is evaluated on various metrics like MSE, Conflict ratio, failure ratio, Throughput, CPU utilization and execution time. The performance is evaluated across varying transaction volumes, including 25, 50, 75, and 100 transactions of the system. MSE value fluctuates across different transaction counts, indicating varying degrees of accuracy in predicting outcomes. The MSE values range from 0.0000002948 to 0.0000052466, with relatively low prediction errors overall transaction. Throughput, measured in Mbps (Megabits per second), we observe a consistent trend of increasing throughput with a higher number of transactions. This indicates that the system can transmit data at a higher rate as the workload intensifies. The conflict and failure ratios remain relatively stable across different transaction counts, indicating consistent performance in managing conflicts and transaction failures. There is a slight variation in execution time, with values ranging from 0.04 to 0.054 seconds, indicating minor fluctuations in the time taken to complete transactions. CPU utilization remains relatively high, consistently above 86%, indicating efficient utilization of system resources across varying transaction loads. The analysis highlights the system's capability to maintain stable performance across different transaction volumes.

The result of the proposed BPNN-BSSA model is compared to analyse the performance based on various metrics. The BPNN-BSSA model outperforms other models in terms of MSE, showcasing a significantly lower value compared to other methodologies. The MSE value of BPNN-BSSA is 0.00000275 which is higher than that of HCC-MGSA-ART (0.00000101), HBOCC (0.00000094), SAOL –LRCD (0.00000114), RDMA-RCC (0.0000007), TCC –TDG (0.00000112), ES2PL (0.00000081), MCSI-NVM (0.00000098), DPC2-CD (0.00000037), RDMA-2PL (0.00000044), RDMA (0.00000068), and TEE –BCCH (0.00000092). The model demonstrates higher throughput value of BPNN-BSSA of 3.523 Mbps is lower than that 1.349mbps, 2.158mbps, 1.044 Mbps, 2.086 Mbps, 1.536 Mbps, 0.886 Mbps, 2.123 Mbps, 0.497 Mbps, 1.764 Mbps, 0.9114 Mbps, 2.096 Mbps, lower than HCC-MGSA-ART, HBOCC, SAOL –LRCD, RDMA-RCC, TCC –TDG, ES2PL, MCSI-NVM, DPC2-CD, RDMA-2PL, RDMA and TEE –BCCH methods. The BPNN-BSSA model exhibits a lower conflict ratio compared to other models. The conflict ratio of BPNN-BSSA (0.42) is higher and has a lower failure ratio (0.05) compared to HCC-MGSA-ART, HBOCC, SAOL –LRCD, RDMA-

RCC, TCC –TDG, ES2PL, MCSI-NVM, DPC2-CD, RDMA-2PL, RDMA and TEE –BCCH methods. The resultants signify that the BPNN-BSSA based on reliability for transaction processing enhanced the performance in the distributed environment.

4.4 CHAPTER SUMMARY

This chapter discusses the experimental setup, performance evaluation and the results of the proposed model and also the results are compared to the recent methods for transaction processing. The comparative analysis showed that the BPNN-BSSA model demonstrated superior performance and provided higher accuracy for prediction, better synchronization and coordination, higher reliability, and overall effectiveness for transaction processing systems. The model handled non-linear relationship and complexity problems between the I/O with efficient performance and enhanced the concurrency control in the database. Overall, the proposed BPNN-BSSA model showed a promising solution for scalable and efficient transaction management in the distributed database environment.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. CONCLUSION

The hybrid BPNN-BSSA model effectively manages the intricate and non-linear correlation between input and output variables, demonstrating efficient performance. By identifying the optimal set of weight and bias parameters, the model minimizes error functions and enhances concurrency control efficiency. Moreover, it dynamically selects between optimistic and pessimistic concurrency models based on the observed conflict rate, adapting locking strategies according to the workload of the database. Through the integration of BPNN and BSSA, the model proficiently handles the transaction and versioning mechanisms, thereby ensuring and augmenting concurrency control within the database. This innovative approach, merging neural networks with optimization techniques, successfully tackles the challenges associated with concurrent resource access in database systems.

The model demonstrated exceptional performance compared to existing methods, showcasing lower MSE, increased throughput, and minimized conflict and failure ratios. The selection of concurrency control strategy dynamically adjusts based on the conflict rate, allowing the model to choose an appropriate locking strategy tailored to current database workloads. Integration of mechanisms for both committed and uncommitted transactions further enhances the model's capability for robust concurrency control. By adapting to different concurrency control approaches and utilizing optimized neural network parameters, the model emerges as a promising solution for efficient and scalable transaction management in modern distributed database environments.

5.2 FUTURE WORK

The research work can be further continued in future with the following methods,

- The current model focuses on transaction processing in a general context. The model could explore the applicability in real-time transaction processing scenarios, where low latency and high throughput are critical requirements.

- The model can be emerged with blockchain-based databases or quantum databases, future work could investigate how the BPNN-BSSA model could be adapted or extended to support these emerging technologies and their unique transaction management requirements.
- To facilitate wider adoption and usability the model could involve developing user-friendly interfaces, tools, or frameworks that simplify the deployment, configuration, and management of the model in various database environments.
- To ensure protection it is essential to incorporate techniques such as differential privacy and homomorphic encryption to safeguard sensitive data and privacy during the optimization process. This ensures that user data confidentiality is maintained, even during collaborative decision-making.

REFERENCES

[1] Abba Ari, A. A., Djedouboum, A. C., Gueroui, A. M., Thiare, O., Mohamadou, A., & Aliouat, Z. (2020). A three-tier architecture of large-scale wireless sensor networks for big data collection. Applied Sciences, 10(15), 5382.

[2] Abdelhafiz, B. M. (2020, December). Distributed database using sharding database architecture. In 2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE) (pp. 1-17). IEEE.

[3] Abduljalil, E., Thabit, F., Can, O., Patil, P. R., & Thorat, S. B. (2022). A new secure 2PL real-time concurrency control algorithm (ES2PL). International Journal of Intelligent Networks, *3*, 48-57.

[4] Abuya, T. K., Rimiru, R. M., & Cheruiyot, W. K. A failure recovery algorithm in Two-Phase commit protocol for optimizing transaction atomicity.

[5] Ali, M. G. (2022). Adopting a Proxy Database to Prevent Direct Access to Distributed Transaction Databases Ensuring Information Security. Journal of Advances in Mathematics and Computer Science, *37*(3), 43-55.

[6] Ali, S., Alauldeen, R., & Ruaa, A. (2020). What is Client-Server System: Architecture, Issues and Challenge of Client-Server System. HBRP Publication, 2(1), 1-6.

[7] Al-Qerem, A., Alauthman, M., Almomani, A., & Gupta, B. B. (2020). IoT transaction processing through cooperative concurrency control on fog–cloud computing environment. Soft Computing, *24*, 5695-5711.

[8] Arora, V., Bhatia, R., & Singh, M. (2016). A systematic review of approaches for testing concurrent programs. Concurrency and Computation: Practice and Experience, 28(5), 1572-1611.

[9] Bernstein, P. A., Das, S., Ding, B., & Pilman, M. (2015, May). Optimizing optimistic concurrency control for tree-structured, log-structured databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (pp. 1295-1309).

[10] Bharati, R., & Attar, V. (2023). Hybrid Graph Partitioning with OLB Approach in Distributed Transactions. Intelligent Automation & Soft Computing, *37*(1).

[11] Chaudhry, N., & Yousaf, M. M. (2022). Concurrency control for real-time and mobile transactions: Historical view, challenges, and evolution of practices. Concurrency and Computation: Practice and Experience, 34(3), e6549.

[12] Chittayasothorn, S. (2022, June). The Misconception of Relational Database and the ACID Properties. In 2022 8th International Conference on Engineering, Applied Sciences, and Technology (ICEAST) (pp. 30-33). IEEE.

[13] Choi, D., & Song, S. (2016). Concurrency control method to provide transactional processing for cloud data management system. International Journal of Contents, *12*(1), 60-64.

[14] Coronel, C., & Morris, S. (2019). Database systems: design, implementation and management. Cengage learning.

[15] Di Sanzo, P., & Quaglia, F. (2022). On the effects of transaction data access patterns on performance in lock-based concurrency control. IEEE Transactions on Computers.

[16] Diallo, O., Rodrigues, J. J., Sene, M., & Lloret, J. (2013). Distributed database management techniques for wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems, 26(2), 604-620.

[17] Dickerson, T., Gazzillo, P., Herlihy, M., & Koskinen, E. (2017, July). Adding concurrency to smart contracts. In Proceedings of the ACM Symposium on Principles of Distributed Computing (pp. 303-312).

[18] Ding, B., Kot, L., & Gehrke, J. (2018). Improving optimistic concurrency control through transaction batching and operation reordering. Proceedings of the VLDB Endowment, *12*(2), 169-182.

[19] Dinh, T. T. A., Liu, R., Zhang, M., Chen, G., Ooi, B. C., & Wang, J. (2018). Untangling blockchain: A data processing view of blockchain systems. IEEE transactions on knowledge and data engineering, 30(7), 1366-1385.

[20] Diop, L., Diop, C. T., Giacometti, A., & Soulet, A. (2022). Pattern on demand in transactional distributed databases. Information Systems, *104*, 101908.

[21] Dizdarevic, J., Avdagic, Z., Orucevic, F., & Omanovic, S. (2021). Advanced consistency management of highly-distributed transactional database in a hybrid cloud environment using novel R-TBC/RTA approach. Journal of Cloud Computing, 10, 1-31.

[22] Dong, Z. Y., Tang, C. Z., Wang, J. C., Wang, Z. G., Chen, H. B., & Zang, B. Y. (2020). Optimistic transaction processing in deterministic database. Journal of Computer Science and Technology, 35, 382-394.

[23] Fan, P., Liu, J., Yin, W., Wang, H., Chen, X., & Sun, H. (2020). 2PC*: a distributed transaction concurrency control protocol of multi-microservice based on cloud computing platform. Journal of Cloud Computing, 9, 1-22.

[24] Freitag, M., Kemper, A., & Neumann, T. (2022). Memory-optimized multi-version concurrency control for disk-based database systems. Proceedings of the VLDB Endowment, 15(11), 2797-2810.

[25] Gabriel, B. C., & Kabari, L. G. (2020). Hybridized concurrency control technique for transaction processing in distributed database system. International Journal of Computer Science and Mobile Computing, 9(9), 118-127.

[26] Gharaibeh, A., Salahuddin, M. A., Hussini, S. J., Khreishah, A., Khalil, I., Guizani, M., & Al-Fuqaha, A. (2017). Smart cities: A survey on data management, security, and enabling technologies. IEEE Communications Surveys & Tutorials, 19(4), 2456-2501.

[27] Ghasemi-Marzbali, A. (2020). A novel nature-inspired meta-heuristic algorithm for optimization: bear smell search algorithm. Soft computing, 24(17), 13003-13035.

[28] Gohil, J. A., & Dolia, P. M. (2016). Design, implementation and performance analysis of concurrency control algorithm with architecture for temporal database. International Journal of Database Management Systems (IJDMS), 8(5), 25-38.

[29] Gohil, J. A., & Dolia, P. M. (2016). Graphical Representation of Optimistic Locking and Concurrency Control for Temporal Database using

Oracle 12c Enterprise Manager. International Journal of Computer Applications, 148(7).

[30] Grohmann, J., Seybold, D., Eismann, S., Leznik, M., Kounev, S., & Domaschka, J. (2020, November). Baloo: Measuring and modeling the performance configurations of distributed dbms. In 2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS) (pp. 1-8). IEEE.

[31] Guo, J., Cai, P., Wang, J., Qian, W., & Zhou, A. (2019). Adaptive optimistic concurrency control for heterogeneous workloads. Proceedings of the VLDB Endowment, 12(5), 584-596.

[32] Guo, T., Zhang, Z., Yuan, Y., Yang, X., & Wang, G. (2024). Hybrid concurrency control protocol for data sharing among heterogeneous blockchains. Frontiers of Computer Science, 18(3), 183104.

[33] Gupta, S., & Sadoghi, M. (2018, March). EasyCommit: A Non-blocking Two-phase Commit Protocol. In EDBT (pp. 157-168).

[34] Harding, R., Van Aken, D., Pavlo, A., & Stonebraker, M. (2017). An evaluation of distributed concurrency control. Proceedings of the VLDB Endowment, 10(5), 553-564.

[35] Jepsen, T., de Sousa, L. P., Moshref, M., Pedone, F., & Soulé, R. (2018, August). Infinite resources for optimistic concurrency control. In Proceedings of the 2018 Morning Workshop on In-Network Computing (pp. 26-32).

[36] Jin, C., Pang, S., Qi, X., Zhang, Z., & Zhou, A. (2021). A high performance concurrency protocol for smart contracts of permissioned blockchain. IEEE Transactions on Knowledge and Data Engineering, *34*(11), 5070-5083.

[37] Jingyao, L. I., Qian, Z. H. A. N. G., Zhanhao, Z. H. A. O., Wei, L. U., Xiao, Z. H. A. N. G., & Xiaoyong, D. U. (2023). RDMA Optimization Technology for Two-Phase Locking Concurrency Control. Journal of Frontiers of Computer Science & Technology, 17(5), 1201.

[38] Jun, W., & Hong, S. K. (2018). Development and performance evaluation of a concurrency control technique in object-oriented database

systems. KSII Transactions on Internet and Information Systems (TIIS), 12(4), 1899-1911.

[39] Kambayashi, T., Tanabe, T., Hoshino, T., & Kawashima, H. (2023). Shirakami: A hybrid concurrency control protocol for tsurugi relational database system. arXiv preprint arXiv:2303.18142.

[40] Kniep, Q., Kokoris-Kogias, L., Sonnino, A., Zablotchi, I., & Zhang, N. (2024). Pilotfish: Distributed Transaction Execution for Lazy Blockchains. arXiv preprint arXiv:2401.16292.

[41] Li, Y., Chen, Z., Cai, Y., Huang, D., & Li, Q. (2017). Accelerating convolutional neural networks using fine-tuned backpropagation progress. In Database Systems for Advanced Applications: DASFAA 2017 International Workshops: BDMS, BDQM, SeCoP, and DMMOOC, Suzhou, China, March 27-30, 2017, Proceedings 22 (pp. 256-266). Springer International Publishing.

[42] Lin, Q., Chen, G., & Zhang, M. (2018, April). On the design of adaptive and speculative concurrency control in distributed databases. In 2018 IEEE 34th International Conference on Data Engineering (ICDE) (pp. 1376-1379). IEEE.

[43] Liu, J., Zhang, S., Wang, Q., & Wei, J. (2022). Coordinating fast concurrency adapting with autoscaling for slo-oriented web applications. IEEE Transactions on Parallel and Distributed Systems, 33(12), 3349-3362.

[44] Liu, X., Chen, K., Liu, M., Cai, S., Wu, Y., & Zheng, W. (2022). Multi-Clock Snapshot Isolation Concurrency Control for NVM Database. Tsinghua Science and Technology, 27(6), 925-938.

[45] Lokhande, D. B., & Dhainje, P. B. (2019). A novel approach for transaction management in heterogeneous distributed real time replicated database systems. International Journal for Scientific Research and Development, 7(1), 840-844.

[46] Lotfy, A. E., Saleh, A. I., El-Ghareeb, H. A., & Ali, H. A. (2016). A middle layer solution to support ACID properties for NoSQL databases. Journal of King Saud University-Computer and Information Sciences, 28(1), 133-145.

[47] Lyu, Z., Zhang, H. H., Xiong, G., Guo, G., Wang, H., Chen, J., ... & Raghavan, V. (2021, June). Greenplum: a hybrid database for transactional and analytical workloads. In Proceedings of the 2021 International Conference on Management of Data (pp. 2530-2542).

[48] Mahajan, S., & Jain, L. (2023). A Study of Distributed Database Management System. International Journal of Engineering and Management Research, 13(2), 121-126.

[49] Masumura, K., Hoshino, T., & Kawashima, H. (2022). Fast concurrency control with thread activity management beyond backoff. Journal of Information Processing, 30, 552-561.

[50] Mohamed, M., Badawy, M., & EL-Sayed, A. (2019). An improved algorithm for database concurrency control. International Journal of Information Technology, 11, 21-30.

[51] Mohana, M., & Jaykumar, C. (2017). Hierarchical replication and multiversion concurrency control model for mobile database systems (MDS). *Wireless Networks*, *23*, 1401-1411.

[52] Moiz, S. (2015). A hybrid concurrency control strategy for mobile database systems. Advances in Research, 3(4), 374-381.

[53] Muslihah, I., & Nastura, S. A. (2020). Transaction Processing System Analysis Using The Distribution Management System (DMS) Nexsoft Distribution 6 (ND6). International Journal of Computer and Information System (IJCIS), 1(2), 31-34.

[54] Nakamura, Y., Kawashima, H., & Tatebe, O. (2019). Integration of tictoc concurrency control protocol with parallel write ahead logging protocol. International Journal of Networking and Computing, 9(2), 339-353.

[55] Neumann, T., Mühlbauer, T., & Kemper, A. (2015, May). Fast serializable multi-version concurrency control for main-memory database systems. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (pp. 677-689).

[56] Nguyen, T., & Kawashima, H. (2023, November). Fairly decentralizing a hybrid concurrency control protocol for real-time database

systems. In 2023 Eleventh International Symposium on Computing and Networking Workshops (CANDARW) (pp. 24-30). IEEE.

[57] Nikolić, L., Dimitrieski, V., & Čeliković, M. (2024). An approach for supporting transparent acid transactions over heterogeneous data stores in microservice architectures. Computer Science and Information Systems, (00), 6-6.

[58] Olivares-Rojas, J. C., Reyes-Archundia, E., Gutiérrez-Gnecchi, J. A., González-Murueta, J. W., & Cerda-Jacobo, J. (2020). A multi-tier architecture for data analytics in smart metering systems. Simulation Modelling Practice and Theory, 102, 102024.

[59] Özsu, M. T., & Valduriez, P. (1999). Principles of distributed database systems (Vol. 2). Englewood Cliffs: Prentice Hall.

[60] Pan, H., Ta, Q. T., Zhang, M., Chee, Y. M., Chen, G., & Ooi, B. C. (2023). FLAC: A Robust Failure-Aware Atomic Commit Protocol for Distributed Transactions. arXiv preprint arXiv:2302.04500.

[61] Pandey, S., & Shanker, U. (2018). On using priority inheritance-based distributed static two-phase locking protocol. In Advances in Data and Information Sciences: Proceedings of ICDIS-2017, Volume 1 (pp. 179-188). Springer Singapore.

[62] Pandey, S., & Shanker, U. (2018, June). CART: a real-time concurrency control protocol. In Proceedings of the 22nd International Database Engineering & Applications Symposium (pp. 119-128).

[63] Poudel, P., Rai, S., & Guragain, S. (2024). Ordered scheduling in control-flow distributed transactional memory. Theoretical Computer Science, 114463.

[64] Ramesh, D., Gupta, H., Singh, K., & Kumar, C. (2015). Hash based incremental optimistic concurrency control algorithm in distributed databases. In Intelligent Distributed Computing (pp. 139-150). Springer International Publishing.

[65] Sheikhan, M., & Ahmadluei, S. (2013). An intelligent hybrid optimistic/pessimistic concurrency control algorithm for centralized database systems using modified GSA-optimized ART neural model. Neural Computing and Applications, 23, 1815-1829.

[66] Sheikhan, M., Rohani, M., & Ahmadluei, S. (2013). A neural-based concurrency control algorithm for database systems. Neural Computing and Applications, 22, 161-174.

[67] Shen, S., Wei, X., Chen, R., Chen, H., & Zang, B. (2022). DrTM+ B: Replication-driven live reconfiguration for fast and general distributed transaction processing. *IEEE Transactions on Parallel and Distributed Systems*, *33*(10), 2628-2643.

[68] Stephan, J., Fink, J., Kumar, V., & Ribeiro, A. (2017). Concurrent control of mobility and communication in multirobot systems. IEEE Transactions on Robotics, 33(5), 1248-1254.

[69] Taft, R., Mansour, E., Serafini, M., Duggan, J., Elmore, A. J., Aboulnaga, A., ... & Stonebraker, M. (2014). E-store: Fine-grained elastic partitioning for distributed transaction processing systems. Proceedings of the VLDB Endowment, 8(3), 245-256.

[70] Uchida, H., & Kawashima, H. (2024). CLMD: Making Lock Manager Predictable and Concurrent for Deterministic Concurrency Control. International Journal of Networking and Computing, 14(1), 81-92.

[71] Wang, C., & Qian, X. (2021). RDMA-enabled concurrency control protocols for transactions in the cloud era. *IEEE Transactions on Cloud Computing*, *11*(1), 798-810.

[72] Wang, Q., Chen, H., Zhang, S., Hu, L., & Palanisamy, B. (2018). Integrating concurrency control in n-tier application scaling management in the cloud. IEEE Transactions on Parallel and Distributed Systems, 30(4), 855-869.

[73] Wang, T., & Kimura, H. (2016). Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores. Proceedings of the VLDB Endowment, 10(2), 49-60.

[74] Wang, T., Johnson, R., Fekete, A., & Pandis, I. (2017). Efficiently making (almost) any concurrency control mechanism serializable. The VLDB Journal, 26, 537-562.

[75] Wei, X., Dong, Z., Chen, R., & Chen, H. (2018). Deconstructing {RDMA-enabled} distributed transactions: Hybrid is better!. In 13th

USENIX Symposium on Operating Systems Design and Implementation (OSDI 18) (pp. 233-251).

[76] Wei, X., Shi, J., Chen, Y., Chen, R., & Chen, H. (2015, October). Fast in-memory transaction processing using RDMA and HTM. In Proceedings of the 25th Symposium on Operating Systems Principles (pp. 87-104).

[77] Weng, S., Wang, Q., Qu, L., Zhang, R., Cai, P., Qian, W., & Zhou, A. (2024). Lauca: A Workload Duplicator for Benchmarking Transactional Database Performance. IEEE Transactions on Knowledge and Data Engineering.

[78] Wu, Y., Arulraj, J., Lin, J., Xian, R., & Pavlo, A. (2017). An empirical evaluation of in-memory multi-version concurrency control. Proceedings of the VLDB Endowment, 10(7), 781-792.

[79] Xia, H., Chen, J., Ma, N., Huang, J., & Du, X. (2023, April). Efficient execution of blockchain transactions through deterministic concurrency control. In International Conference on Database Systems for Advanced Applications (pp. 509-518). Cham: Springer Nature Switzerland.

[80] Xia, Y., Yu, X., Butrovich, M., Pavlo, A., & Devadas, S. (2022, June). Litmus: Towards a practical database management system with verifiable acid properties and transaction correctness. In Proceedings of the 2022 international conference on management of data (pp. 1478-1492).

[81] Yadav, A. K., Raw, R. S., & Bharti, R. K. (2023). DPC 2-CD: a secure architecture and methods for distributed processing and concurrency control in cloud databases. Cluster Computing, 26(3), 2047-2068.

[82] Yamada, H., & Nemoto, J. (2022). Scalar DL: Scalable and practical Byzantine fault detection for transactional database systems. Proceedings of the VLDB Endowment, 15(7), 1324-1336.

[83] Yao, C., Agrawal, D., Chang, P., Chen, G., Ooi, B. C., Wong, W. F., & Zhang, M. (2015). Dgcc: A new dependency graph based concurrency control protocol for multicore database systems. arXiv preprint arXiv:1503.03642.

[84] Yao, C., Zhang, M., Lin, Q., Ooi, B. C., & Xu, J. (2018). Scaling distributed transaction processing and recovery based on dependency logging. The VLDB Journal, *27*, 347-368.

[85] Yu, X., Pavlo, A., Sanchez, D., & Devadas, S. (2016, June). Tictoc: Time traveling optimistic concurrency control. In Proceedings of the 2016 International Conference on Management of Data (pp. 1629-1642).

[86] Yu, X., Xia, Y., Pavlo, A., Sanchez, D., Rudolph, L., & Devadas, S. (2018). Sundial: Harmonizing concurrency control and caching in a distributed OLTP database management system. Proceedings of the VLDB Endowment, 11(10), 1289-1302.

[87] Zhang, Q., Li, J., Zhao, H., Xu, Q., Lu, W., Xiao, J., ... & Du, X. (2023). Efficient distributed transaction processing in heterogeneous networks. Proceedings of the VLDB Endowment, 16(6), 1372-1385.

[88] Zhao, H., Li, J., Lu, W., Zhang, Q., Yang, W., Zhong, J., ... & Pan, A. (2024). RCBench: an RDMA-enabled transaction framework for analyzing concurrency control algorithms. The VLDB Journal, 33(2), 543-567.

[89] Žužek, T., Kušar, J., Rihar, L., & Berlec, T. (2020). Agile-Concurrent hybrid: A framework for concurrent product development using Scrum. Concurrent Engineering, 28(4), 255-264.

جمهورية العراق

وزارة التعليم العالي والبحث العلمي

جامعة تكريت

كلية علوم الحاسوب والرياضيات

قسم علوم الحاسوب

تطوير نموذج التحكم في التزامن الهجين لمعالجة المعاملات في نظام قاعدة البيانات الموزعة

رسالة مقدمة إلى

مجلس كلية علوم الحاسوب والرياضيات،

جامعة تكريت،

كجزء من متطلبات الحصول على درجة الماجستير في علوم الحاسوب

الطالب

مثنى عبدالله صالح

بإشراف

أ.م.د سعدي حمد ثلج

1446هـ 2024 م

ملخص

بيان المشكلة: في عالم الأنظمة الموزعة المترابط، أصبحت أنظمة إدارة قواعد البيانات الموزعة (DDBMS) بالغة الأهمية لإدارة البيانات المنتشرة عبر مواقع متعددة. تواجه أنظمة إدارة قواعد البيانات الموزعة تحديات في التحكم في التزامن، حيث تعد إدارة العمليات من خلال تنفيذ المعاملات في وقت واحد أمرًا بالغ الأهمية. يمكن أن تعاني نماذج التحكم في التزامن المتفائلة من مشكلات في الأداء بسبب معدلات التعارض المتفاوتة بين المعاملات. هناك حاجة إلى نهج ديناميكي لاختيار أفضل استراتيجية للتحكم في التزامن بشكل تكيفي بناءً على ظروف المعاملات في الوقت الفعلي.

الهدف: الهدف من هذه الدراسة هو تطوير نموذج تطوير يختار بشكل ديناميكي بين أساليب التحكم في التزامن المتفائلة والمتشائمة في أنظمة إدارة قواعد البيانات الموزعة. تهدف هذه الدراسة إلى تطوير خوارزمية تتحول بشكل ديناميكي بين التحكم في التزامن المتفائل والمتشائم في أنظمة إدارة قواعد البيانات الموزعة، وتحسين الأداء والاتساق من خلال التكيف مع معدلات التعارض المتفاوتة.

المنهجية: تستفيد الخوارزمية المقترحة من شبكة عصبية ذات انتشار خلفي (BPNN) لتحديد ما إذا كان سيتم منح قفل أو تحديد المعاملة الفائزة في بيئة قاعدة بيانات موزعة. تم تحسين معلمات

BPNN باستخدام خوارزمية البحث عن رائحة الدب (BSSA)، والتي تم تصميمها لضبط الشبكة بشكل دقيق من أجل اتخاذ قرارات أفضل. مع إدراك أنه في الإعدادات العملية، غالبًا ما تكون المعلومات حول مجموعة الكتابة (WS) ومجموعة القراءة (RS) متاحة جزئيًا فقط قبل تنفيذ المعاملة، تم تصميم الخوارزمية لاستيعاب السيناريوهات التي تحتوي على بيانات WS وRS اختيارية أو غير كاملة. تسمح آلية الاختيار الديناميكية هذه، المدعومة بمجموعة BPNN-BSSA، للنظام باختيار أفضل استراتيجية للتحكم في التزامن بشكل تكيفي بناءً على معدل التعارض الحالي.

النتائج: أظهر تنفيذ خوارزمية BSSA-BPNN تحسينات كبيرة في كل من معدل إنتاج المعاملات والاتساق مقارنة بأساليب التحكم في التزامن الثابتة التقليدية. من خلال التعديل الديناميكي لمعدل التعارض في الوقت الفعلي، تمكنت الخوارزمية من تقليل تعارضات المعاملات وتحسين الأداء العام للنظام، وخاصة في سيناريوهات التعارض العالي. الاستنتاج: توفر خوارزمية BSSA-BPNN المقترحة حلاً فعالاً للتحكم الديناميكي في التزامن في أنظمة إدارة قواعد البيانات الموزعة، وموازنة الأداء والاتساق من خلال التكيف مع معدلات التعارض المتنوعة. يعزز دمج تقنيات التعلم الآلي والتحسين في عملية التحكم في التزامن قدرة النظام على التعامل مع ظروف المعاملات المتنوعة، مما يجعله خيارًا قويًا لبيئات قواعد البيانات الموزعة الحديثة.

القيـود والعمـل المسـتقبلي: فـي حـين أن خوارزميـة BSSA-BPNN واعـدة، إلا أنهـا ليسـت خاليـة مـن القيـود. تعتمـد فعاليـة الخوارزميـة علـى دقـة التنبـؤ بمعـدل التعـارض، والـذي يمكـن أن يتـأثر بـالتغيرات غيـر المتوقعـة فـي أنمـاط المعـاملات. بالإضـافة إلـى ذلـك، قـد تـؤثر النفقـات الحسـابية التـي تقـدمها BPNN وBSSA علـى الأداء فـي الأنظمـة ذات أحجـام المعـاملات المرتفعـة للغايـة. يجـب أن يركـز العمـل المسـتقبلي علـى تحسـين آليـة التنبـؤ بمعـدل التعـارض، واستكشـاف خوارزميـات التحسـين البديلـة، واختبـار الخوارزميـة فـي بيئـات موزعـة أكثـر تنوعًـا وأكبـر نطاقًـا لتعزيـز قابليـة التوسـع والكفاءة بشكل أكبر.