Analysis on Replicated Concurrency Control and Recovery in Distributed Database Systems

Sanjida Ali Shusmita
Department of Computer Science and Engineering
BRAC University
66 Mohakhali, Dhaka-1212, Bangladesh
sanjida.ali.shusmita@g.bracu.ac.bd

Md. Ashekur Rahman
Department of Computer Science and Engineering
BRAC University
66 Mohakhali, Dhaka-1212, Bangladesh
mohammad.ashekur.rahman@g.bracu.ac.bd

Asma Ul Hussna
Department of Computer Science and Engineering
BRAC University
66 Mohakhali, Dhaka-1212, Bangladesh
asma.ul.hussna@g.bracu.ac.bd

Samiha Khan
Department of Computer Science and Engineering
BRAC University
66 Mohakhali, Dhaka-1212, Bangladesh
samiha.khan@g.bracu.ac.bd

Iffat Immami Trisha
Department of Computer Science and Engineering
BRAC University
66 Mohakhali, Dhaka-1212, Bangladesh
iffat.immami.trisha@g.bracu.ac.bd

Md Sabbir Hossain
Department of Computer Science and Engineering
BRAC University
66 Mohakhali, Dhaka-1212, Bangladesh
md.sabbir.hossain1@g.bracu.ac.bd

Annajiat Alim Rasel
Department of Computer Science and Engineering
BRAC University
66 Mohakhali, Dhaka-1212, Bangladesh
annajiat@gmail.com

***Abstract*—In a fully automated flexible database system, deadlocks are a terrible sign. Their occurrences often degrade resource usage and may result in disastrous outcomes in safety-critical systems. By analyzing the ideas and techniques involved in preventing, avoiding, and detecting deadlocks, this study examines state-of-the-art deadlock control solutions for database management systems. Due to a wide and ongoing stream of activities, the major emphasis of this research is deadlock prevention. This thorough research offers readers with a compilation of the most recent findings in this field, assisting engineers in determining the best method for their industrial application cases. The findings of this study can help to improve future research in this sector, ensuring improved concurrency and resolving complicated concurrency issues.**

***Index Terms*—Distributed Database, Deadlock , Concurrency, Conflict**

I. Introduction

Data and, in many cases, data control are split over two or more physically different locations in distributed data management systems (DDBMS). It could be stored on several computers in a single physical place, such as a data center or it could be distributed across a network of interconnected computers. It is a logically connected collection of data that is technically spread over a computer network or a system that combines a variety of sites connected by a communications network. DDBMS is a combination of several loosely coupled sites with no physical components in general. Here all sites can collaborate such that a user at any site can access data wherever on the network, even if the data is stored at the user's own site and appears to the end user as a single centralized database. A distributed database can be hosted on organized network servers or decentralized independent machines. It is divided into two groups: homogeneous and heterogeneous. To keep its records up to date, it uses replication and duplication among its various sub-databases. All its several database files are managed by a single database management system. Whereas, network connectivity is critical for centralized databases. The longer the database access time is required, the slower the internet connection is. This often leads to bottleneck, when there is high traffic. Distributed databases, on the other hand, rely on a central database management system (DBMS) to manage all of their various storage devices remotely, as they do not need to be stored in the same physical location.

However, in distributed environments, we encounter new techniques such as fragmentation and data replication that do not exist in centralized setups [1]. All the queries and operations on DDBMS sometimes lead to failure as the data is distributed in several sites at the same time. So any kind of data update means it should be done on all of the copies of the database. If one of the sites operation fails that means the whole process needs to be done again. Here we can be introduced with methods like concurrency control, deadlock detection and failure recovery. Concurrency control is a

method of coordinating multiple operations at the same time without them colliding. It ensures that Database transactions are completed in a timely and accurate manner, resulting in accurate results without jeopardizing the database's data security. Also the database management system should detect whether a transaction is in a deadlock or not while it waits indefinitely for a lock.

Here the paper is organized in the following chapters. Rules and requirements of distributed database system are presented in the second chapter. The third section reflects on the problems faced in distributed database system while the fourth section represents database architecture. The fifth chapter presents some related works in this field. The last chapter is the limitation and conclusion of our work.

II. Rules and Requirements of Distributed Database System

Distributed databases benefit from distributed computing, which consists of a large number of processing parts, some of which are heterogeneous. Elements are linked together by a network, which aids in the completion of the prescribed task. It is mostly used to solve problems by breaking them down into smaller chunks and solving them in a coordinated manner. Here we have listed twelve of Date's rule [2]. All of the database management systems may not follow them today, but they give a whole idea of the process. The premise behind these guidelines is that a distributed DBMS should appear to the user to be a non-distributed DBMS. Following are the twelve rules-

Local Autonomy or Local Site Independence: Each site has its own operations and functions as a self-contained, centralized database management system.
Central Site Independence: All of the sites are equal, and none of them rely on the central site to provide any services. Transaction management, query optimization, deadlock detection, and management are some of the functions for which a central server is not required.
Continuous Operation: The system is unaffected by site failure. Even if a site fails or the network expands, the system continues to function.
Local Independence: To retrieve any data from a system, you must first understand data storage, or where the data is kept.
Fragmentation Independence: Only one logical database is shown to the user. Data fragmentation is transparent to the user.
Replication Independence: Data can be replicated and stored at many locations. The DDBMS transparently manages all fragments for the user.
Distributed Query Independence: Any node in the DDBMS that possesses data relevant to the query should be able to perform the query. Many nodes may make contributions to the response to the user's query without the user being aware of it.
Distributed Transaction Independence: While transactions can update data across several locations in a transparent manner, agents are used to govern recovery and concurrency. It may update data across many sites, and it is carried out in a transparent manner.
Hardware Independence: When appropriate, the distributed database system can run on any hardware platform.
Operating System Independence: DDBMS should be able to run on a variety of operating system platforms.
Network Independence: Any network platform can be used to run the DDBMS system. It should be able to run on any network platform that is suitable.
Database Independence: Any database product provider must be supported by the system. That is, assuming the databases have the same interfaces, the DDBS should be able to operate with them.

From the rules it can be stated that, DDBMS is a collection of logically connected shared data that has been divided into several fragments [3]. Fragments may be replicated and they are allocated to sites. Each site's data is managed by a database management system (DBMS), which may handle local applications independently. And it should also maintain transparency. Lastly, Hardware, operating systems, network software, and database management systems are all independent of the system.

III. Concurrency Control

In DDBS, many clients can get to the database concurrently where each clients assumes that he/she is functioning alone on devoted system but usually not the case in DDBS. In that instance, serializable schedules can be used to ensure database consistency because the result obtained will be equal to one of the serial executions of the exchanges. The utilize of only serial plans limits the degree of concurrency thusly influencing execution [4].

When many transactions assassinate concurrently in an undisciplined or unlimited way, at that point it might lead to a few issues [5].The following are the five types of database concurrency issues:: (i). Temporary Update Problem (ii). Incorrect Summary Problem (iii). Lost Update Problem (iv). Unrepeatable Read Problem (v). Phantom Read Problem . The following are the explanations.

A. Temporary Update Problem:

When a transaction upgrades something and fails, it causes a temporary update or dirty read problem.However, the improved item is used in another transaction, and the item is altered or returned to its original value. [5] Example:

In this case, if transaction 1 fails due to some reason, X will revert to its previous value.However, transaction 2 has already read the wrong X value.

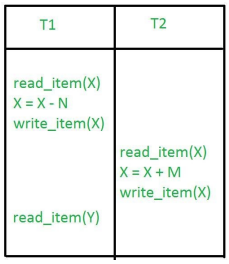

Fig. 1. Temporary Update Problem

B. Incorrect Summary Problem

Consider the following scenario: one transaction applies the total work on a few records, while another transaction upgrades these records.The total work may calculate a few values that have recently been upgraded and some that will be upgraded when they have been upgraded [5]. Example:

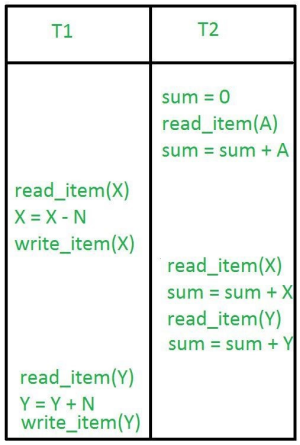

Fig. 2. incorrect Summary Problem

Consider the following scenario: one transaction applies the total work on a few records, while another transaction upgrades these records.The total work may calculate a few values that have recently been upgraded and some that will be upgraded when they have been upgraded.

C. Lost Update Problem

A transaction's upgrade to an information thing is misplaced in the lost update problem because it is overwritten by another transaction's upgrade [5]. Example:

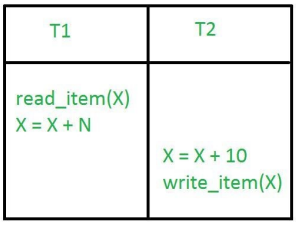

Fig. 3. Lost Update Problem

In this case, transaction 1 modifies the value of X, but this is overridden by transaction 2's update to X.As a result, transaction 1's update is erased.

D. Unrepeatable Read Problem

When two or more read operations in the same transaction read different values of the same variable, the unrepeatable problem occurs. [5] Example:

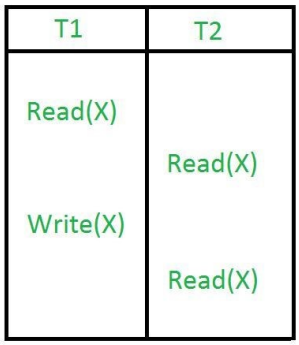

Fig. 4. Unrepeatable Read Problem

After transaction 2 reads the variable X, transaction 1 modifies the value of the variable X with a write operation.As a result, when transaction 2 performs another read operation, it reads the new value of X that was upgraded by transaction 1.

1) Phantom Read Problem : When a transaction reads a variable once, it gets an error saying the variable doesn't exist when it tries to read it again. [5] Example: When transaction

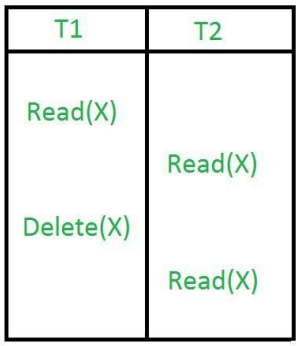

Fig. 5. Phantom Read Problem

2 reads the variable X, transaction 1 deletes the variable X without the knowledge of transaction 1.As a result, when transaction 2 attempts to read X, it is unable to do so.

IV. Deadlock Detection

Deadlock detection is a prominent issue not only in distributed systems [6] but also in databases [7].Under the deadlock detection, deadlocks are tolerated. The state of the system is then reviewed to see if a stalemate has occurred, and if so, it is resolved.The operating system's resource scheduler knows what resources each process has locked and/or is actively

requesting, detecting a deadlock that has already happened is simple. There are two condition in that case occurs: (1) If a resource only has one instance: In this case, we can apply an algorithm to detect deadlock by looking for cycles in the Resource Allocation Graph [8].

Example: There are just one instance of each resource in

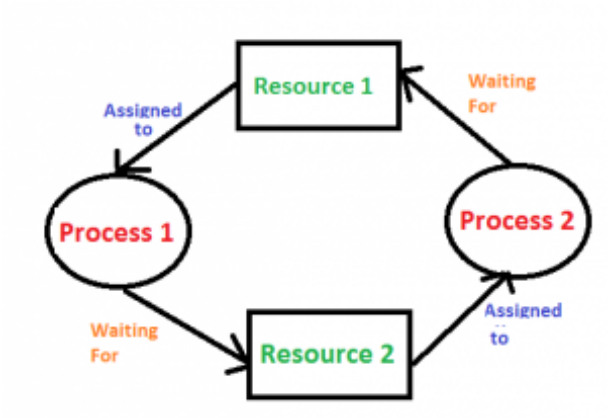

Fig. 6. Deadlock Detection

this figure-6.There is a cycle R1 → P1 → R2 → P2. So, Deadlock has been created. (2)If there are several instances of the same resource: The identification of the cycle is a required but insufficient requirement for the discovery of deadlock; Depending on the situation, the system may or may not be in stalemate in this case. [8]

V. Read Write Transaction

At the start of a READONLY transaction, a logical snapshot of the database is created and then released.This ensures that the database provides consistent data to all reads in all statements throughout this transaction.For example, if a transaction with multiple statements takes 10 hours to complete, a database snapshot for this transaction will be created for 10 hours. When a query statement is run at the start and conclusion of a transaction, the result is guaranteed to be the same.To put it another way, any data modifications made by other users during this 10-hour period will have no bearing on the execution of statements within this transaction. [9] In a database transaction, there are three sorts of conflicts. • Write-Read (WR) conflict • Read-Write (RW) conflict • Write-Write (WW) conflict

A. Write-Read (WR) conflict

When a transaction reads data written by another transaction before committing, it causes a conflict. [9] The following concepts are used in this figure-9: A and B are two separate database data objects. T1 and T2 are two distinct transactions.R(A) - reading data A.W(A) - Writing data A. Com. - committing transaction

Example: Before T2 commits, the transaction T2 is reading the data that T1 has written. Dirty Read is another name for it. It breaks the data consistency rule's ACID property. Expected

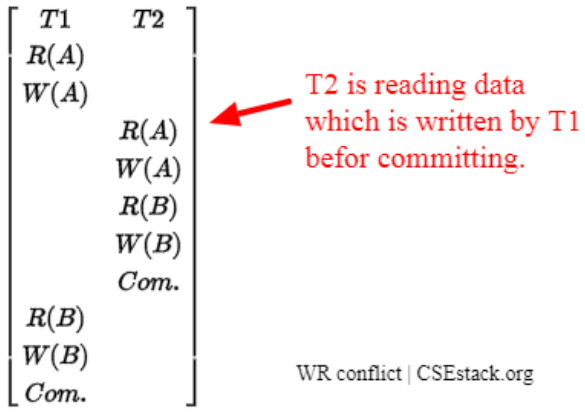

Fig. 7. Write-Read (WR) conflict

behavior: If another transaction (say T1) has already written the data and has not committed, the new transaction (say T2) should not read it.

B. Read-Write (RW) conflict

Example: T2 is a transaction that writes data that T1 has

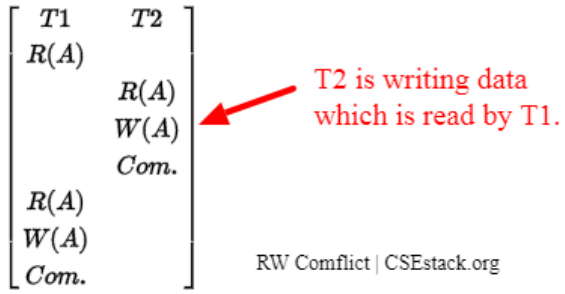

Fig. 8. Read-Write (RW) conflict

already read. You'll see that the data read by transaction T1 before and after T2 commits differs in the diagram above. Example: Let's say Alice and Bob wish to book their vacation flight. Alice goes to the airline's website to check the ticket's availability and price. There is just one available ticket. Alice thinks it's too pricey and decides to look for other airline fares if she receives any. Bob, on the other hand, logs into the same airline website and purchases the ticket. There are currently no more flight tickets available. When Alice is unable to locate a suitable offer on another airline portal, she returns to the prior airline portal. And I tried to get an airline ticket, but it was sold out. Its Conflict, Read-Write (WR) conflict.

C. Write-Write (WW) conflict

Example: Transaction T2 is now writing data that Transaction T1 has previously written. T2 overwrites the data that T1 has written. A blind write operation is another name for it. T1's data is no longer available. As a result, there is a data update loss. Alice and Bob, for example, have a Google-sheet that they share online. Both of them read the document.Bob overwrites the content that Alice has changed. The data that

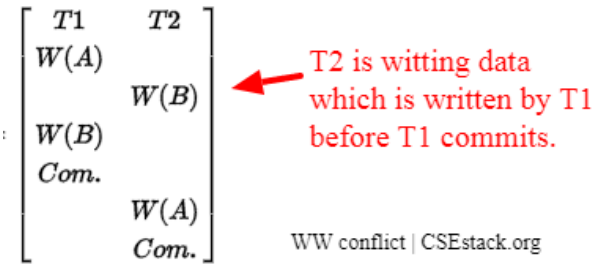

Fig. 9. Write-Write (WW) conflict

Alice had revised had vanished. The disagreement is known as Write-Write (WW) conflict.

VI. Distributed Database Arcitectures

The DDBMS architecture is usually designed based on three parameters::

- Distribution:- Data are physically distributed over the different sites.
- Autonomy :- It denotes the distribution of database system control and the degree to which each part of DBMS can work independently.
- Heterogeneity :- :- It denotes the similarity or distinction of data models, system components, and databases.

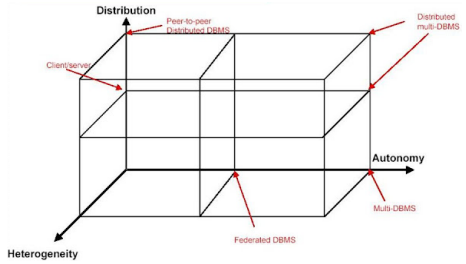

Fig. 10. 3-dimensional architectural model

Architecture Models for DDBMS can be classified along three dimensions:-

- Client server Architecture :- it has divided functionality into client and server where client mostly deals with user interface and server deals with primarily encompass data management, query processing, optimization and transaction management.
- Peer to peer Architecture :- In this system peer acts as both client and server and share resources with each other and co-ordinates.
- Multi DBMS Architecture :- As name suggests it is integrated database system consisting of two or more databases

Concurrency control helps multiple transactions to executed simultaneously. It restricts multiple users from editing the same data at the same time and serializes transactions for backup and recovery purposes. [4]. It also maintains the ACID properties of the transactions. There are mainly 3 types concurrency control techniques: -

- Lock Based Concurrency Control
- Time Stamp Concurrency Control
- Optimistic Concurrency Control

1) Lock Based Concurrency Control: - As name suggested before no transaction can read or write data until it obtains the appropriate lock.
   - Shared Lock :- this is also known as a read-only lock and it can shared between the transactions.
   - Exclusive Lock :- In exclusive lock both read and write is possible in the transaction.

   One phase locking protocol also provide maximum concurrency control. When it locks item for each transaction, it uses and release the lock when work finished. But it doesn't provide serializability.
   One the Popular Lock based Concurrency control is Distributed Two Phase Locking Protocol (2PL) Where it has mainly 2 execution steps for locking.
   -Firstly, the transaction acquires only the necessary locks and does not release any locks. This is called growing phase.
   -In the second phase, the transaction free the locks and cannot obtain new locks. And also called as shrinking phase.Any transaction that follows the two-phase locking protocol guarantees serialization.
   When it comes to distributed concurrency control Distributed Two phase locking is similar to the two-phase locking protocol. However, in a distributed system lock managers are selected from different sites. Lock Manager manages requests to obtain locks from transaction monitors. It helps to co-ordination between lock manager with different sites.in distributed two-phase locking lock manager controls on data items and stored on their local site.

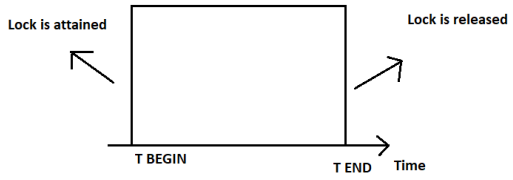

Fig. 11. 2PL Execution diagram

2) Distributed Time Stamp Concurrency Control: -
   Timestamp concurrency control helps the order of transaction based on their timestamps. Older the transaction higher the priority. To determine the timestamp of transaction, it uses system time or logical counter in centralized distributed system. But in Distributed timestamp,for global timestamp any site's local physical or logical clock reading are not allow because it is not

unique. So, Both site ID and site's clock are used as a timestamp. Timestamp based protocols start working as soon as a transaction is created. It also keeps the last timestamp of read and write transactions.

3) Distributed optimistic concurrency control: - it is also known as validation-based concurrency control. It has these phases [10]: - Begin, it will record a timestamp of the start transaction. Read phase where, it will read the value of data items and also stores them temporarily in local variables. it can carry out all the write operations on temporal variables with out updating to the actual database. Validation phase will check temporary variables value with the original value to see if it breaks the serializability. Write phase, if the validation is correct or validated then it will store the value in database if not transaction will roll back.
   In distributed optimistic concurrency control is the mainly two rules :- First rule where,in all sites transaction must be validated locally, if not it should be aborted. And if it is validated it maintains serializability. Second rule, When a transaction locally validated then it should be globally validated as well. if more than two transaction conflict at more than one site,global validation guarantees that they should run in relative order at all the sites.

VII. Deadlock Handling

Multiple users request the same resources from the database. If the resources are usable or obtainable then one user might get that at that time. Other users might wait for the resources to be free. If both users wait indefinitely for resources to be give up by one another then it maybe deadlock [11]. A deadlock can be shown by a cycle in the wait-for-graph. the two major deadlock handling concerns are transaction location and transaction control in a distributed database system . After dealing with these concerns, deadlock can be handled by any of the following approaches There are mainly three approaches for deadlock handling: -

- Distributed Deadlock prevention
- Distributed Deadlock avoidance
- Distributed Deadlock Detection and removal

1) Distributed Deadlock Prevention
   Distributed Deadlock Prevention is similar to Deadlock Prevention where, it will simply doesn't allow any transaction to acquire lock which might lead to deadlock. Popular deadlock prevention methods is pre-acquisition of all the locks. In this method, a transaction need all the locks before starting to execute When transaction enter any of the site controlling site send message to other site where the data items are located to lock the items and wait for confirmation after that retains the locks for the entire duration of transaction. If any site or communication link fails, the transaction has to wait until they have been repaired. if for any reasons the transaction fails then transaction has to wait.

2) Distributed Deadlock Avoidance
   It is just like centralized deadlock avoidance, In this approach, deadlock handle before it occurs. It can easily analyze the transaction and lock whether waiting leads to deadlock or not. Mainly lock manager will check whether the lock available or not. If some incompatible mode means locked by other transaction at that case it uses different algorithms to identify that state causes deadlock or not.That's why distributed nature of the transaction different conflicts may occur –
   - Conflict between two transactions in the same site.
   - Conflict between two transactions in different sites

   There are mainly 2 algorithms for this problem:-
   Distributed Wound-Die:- If transaction1 is older than transaction2, transaction1 is allowed to wait. Transaction1 can resume or start execution after particular site receives a message that transaction2 either committed or aborted .Otherwise, if transaction1 is younger than transaction2, transaction1 is aborted and later restarted. We can see that younger the transaction higher the Priority.
   Distributed Wait-Wait:- If transaction1 is older than transaction2, transaction2 is aborted and later restarted. Transaction2 can resume or start execution after particular site receives a message that transaction1 either committed or aborted . Otherwise, if transaction1 is younger than transaction2, transaction1 is allowed to wait. Here, Older the transaction higher the priority.

3) Distributed Deadlock detection and removal :- Similar to the centralized deadlock detection approach, where it allows to occur deadlock in transaction and try to remove it, if detected. It does not perform any checking mechanism to any transaction before locking. To find deadlocks, the lock manager regularly checks if the wait-for graph has cycles. But, it is difficult to find deadlocks since the transaction is waiting for resources on the network.In distributed deadlock detection algorithms can also use timers.Where each transaction have certain time period in which transaction expected to be finished. And another way to handle deadlock is deadlock detector.
   - Centralized Deadlock Detector
   - Hierarchical Deadlock Detector
   - Distributed Deadlock Detector

VIII. Discussion

The state of the art models for Replicated Concurrency control and recovery techniques are described in Table I.

In this paper, the focus is given on some specific criteria and only relatively recent papers were reviewed to make it not outdated.The motivation explains why they came up with the idea, objective explains the goal what they wanted to do and the main contribution explains what is so new about this new research that did not exist before.

TABLE I
REVIEW OF REPLICATED CONCURRENCY CONTROL AND RECOVERY TECHNIQUES IN DISTRIBUTED DATABASE SYSTEMS

Ref.	Author(s)	Year	Motivation	Objective	Major Contributions
[12]	Szekeres, A., et al.	2020	The authors wanted to identify and resolve distributed database system bottlenecks.	Their objective is to create multicorescalable and replica-scalable replicated systems.	The study describes a novel Zero-Coordination Principle (ZCP) for designing multicore-scalable replicated systems. Meerkat, the first multicore-scalable, replicated, in-memory, transactional storage system, was built on this foundation. The paper did a comparison of Meerkat's performance against that of existing systems to see how breaching ZCP affects multicore scalability and performance.
[13]	Shrivastava, P., & Shanker, U.	2019	There has been no work on predictable real-time transaction (RTT) processing that ensures the mutual consistency of duplicated data.	The authors sought to create a system that could anticipate RTT processing and offer both predictability and mutual consistency, so that it could be used in both present and future real-time applications.	This paper begins by discussing the factors of predictability and mutual consistency in a replicated distributed real-time database system (RDRTDBS), before briefly discussing the features and requirements of RDRTDBS and presenting a processing plan that supports predictable execution of hard, soft, and firm RTT while maintaining mutual consistency. The simulation results show that the suggested processing technique improves RDRTDBS performance above what is currently available from replica update techniques.
[14]	Lu, Y., Yu, X., & Madden, S.	2018	When transactions need to access several partitions, partitioning-based systems suffer, especially in a distributed context where costly protocols like two-phase commit are necessary to coordinate these cross-partition transactions. Non-partitioned methods, on the other hand, perform worse on highly partitionable workloads due to their inability to scale out, but are unaffected by partitionability.	The goal is to combine the advantages of both partitioning and non-partitioning distributed database systems.	STAR can deliver high throughput with serializability guarantees by utilizing multicore parallelism and fast networks. It uses a phase-switching technique that allows STAR to conduct cross-partition transactions without requiring a two-phase commit while maintaining fault tolerance. It employs a hybrid replication method to minimize replication costs while maintaining transactional consistency and high availability.
[15]	Li, J., Michael, E., & Ports, D. R.	2017	To achieve scalability and replication for fault tolerance, a costly mix of atomic commitment and replication protocols was used, resulting in a significant amount of coordination overhead.	Eris is designed to eliminate both replication and transaction coordination overhead as a final outcome.	It integrates multi-sequencing, a key element of concurrency management, into the datacenter network. A new lightweight transaction protocol assures atomicity, and this network primitive is responsible for reliably ordering transactions.
[16]	Ren, K., Li, D., & Abadi, D. J.	2019	On a global scale, applications have been forced to choose between strict serializability, low latency writes, and high transactional throughput.	The objective is to provide a system that manages to avoid this compromise for workloads that need data access from physical regions.	SLOG provides high-throughput, rigorously serializable ACID transactions at geo-replicated distance and scale for all transactions submitted across the globe, while maintaining low latency for transactions that start from a place near to the data they access.Experiments show that compared to existing best models of strictly serializable geo-replicated database systems like Spanner and Calvin, SLOG can cut latency by more than an order of magnitude while retaining good throughput under conflict.

IX. Conclusion

In the realm of research and business, the distributed database is a rapidly growing technology. In this study, we learned about Date's distributed database rules. We also explored many issue areas, methods, and multiple distributed database solutions. Concurrency, deadlock, replication control, security, and privacy can all be easily managed when building a system using the problem areas identified in this study. The architecture of distributed databases in order to construct a distributed database system is also discussed. Finally, several existing works are described, with specifics on particular sectors for each.

References

[1] Y. Wei, A. Aslinger, S. H. Son, and J. A. Stankovic, "Order: A dynamic replication algorithm for periodic transactions in distributed real-time databases," *Department of Computer Science, University of Virginia, USA*. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.5391&rep=rep1&type=pdf

[2] C. J. Date, *Date on Database*. Apress, 2000-2006.

[3] M. Stonebraker and R. Cattell, "10 rules for scalable performance in simple operation' datastores," *Communications of the ACM*, 2011 10.1145/1953122.1953144

[4] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-wesley Reading, 1987, vol. 370. [Online]. Available: http://www.sigmod.org/publications/dblp/db/books/dbtext/bernstein87.html

[5] L. Xavier, "Concurrency problems in dbms transactions," 2019, accessed September 24, 2021. [Online]. Available: https://www.geeksforgeeks.org/concurrency-problems-in-dbms-transactions/

[6] M. Singhal, "Deadlock detection in distributed systems," *Computer*, vol. 22, no. 11, pp. 37–48, 1989 10.1109/2.43525

[7] E. Knapp, "Deadlock detection in distributed databases," *ACM Computing Surveys (CSUR)*, vol. 19, no. 4, pp. 303–328, 1987 10.1145/45075.46163

[8] Vaibhabrai3, "Deadlock detection and recovery," 2021, accessed September 24, 2021. [Online]. Available: https://www.geeksforgeeks.org/deadlock-detection-recovery/

[9] A. Choudhury, "Different types of read write conflict in dbms [explained with example]," 2019, accessed September 24, 2021. [Online]. Available: https://www.csestack.org/different-types-read-write-conflict-database/

[10] H.-T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Transactions on Database Systems (TODS)*, vol. 6, no. 2, pp. 213–226, 1981 10.1145/319566.319567

[11] N. Kaveh and W. Emmerich, "Deadlock detection in distribution object systems," in *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, 2001, pp. 44–51 10.1145/503209.503216

[12] A. Szekeres, M. Whittaker, J. Li, N. K. Sharma, A. Krishnamurthy, D. R. Ports, and I. Zhang, "Meerkat: multicore-scalable replicated transactions following the zero-coordination principle," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–14 10.1145/3342195.3387529

[13] P. Shrivastava and U. Shanker, "Supporting transaction predictability in replicated drtdbs," in *International Conference on Distributed Computing and Internet Technology*. Springer, 2019, pp. 125–140 10.1007/978-3-030-05366-6$_1$0

[14] Y. Lu, X. Yu, and S. Madden, "Star: Scaling transactions through asymmetric replication," *arXiv preprint arXiv:1811.02059*, 2018. [Online]. Available: https://arxiv.org/pdf/1811.02059.pdf

[15] J. Li, E. Michael, and D. R. Ports, "Eris: Coordination-free consistent transactions using in-network concurrency control," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 104–120 10.1145/3132747.3132751

[16] K. Ren, D. Li, and D. J. Abadi, "Slog: Serializable, low-latency, geo-replicated transactions," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1747–1761, 2019 10.14778/3342263.3342647