Distributed Transactions and Distributed Concurrency Control

IJCSMC Journal

IJCSMC

Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 10, Issue. 1, January 2021, pg.61 – 68

Distributed Transactions and Distributed Concurrency Control

Precious B. Gbaranwi[1]; Prince O. Asagba[2]
[1]Department of Computer Science, Ignatius Ajuru University of Education, Nigeria
[1]Email: gbaranwiprecious@yahoo.com
[2]Department of Computer Science, University of Port Harcourt, Nigeria
[2]Email: asagba.prince@uniport.edu.ng
DOI: 10.47760/ijcsmc.2021.v10i01.006

Abstract: Concurrency control is one of the essential tasks of any database management system. It is impracticable to maintain the integrity of the database system in a concurrent environment without concurrency control techniques. This study discusses distributed transactions, ACID properties, and states of transactions, distributed concurrency control and various concurrency control techniques for a distributed database. It also examines the challenges associated with distributed transactions and concurrency control, and highlights the benefits of distributed concurrency control techniques in a distributed environment.
Keywords: Distributed database system, distributed transactions, DDBMS, ACID properties, distributed concurrency control

I. Introduction

In recent years, most of the applications of organizations are dependent on database systems, as they store and maintain the operational data of the organization. Database systems can either be implemented based on centralized or distributed approach. The database system which is implemented based on distributed approach is referred to as distributed database system. The shift from centralized to distributed framework can be attributed to the demand for higher performance, speed and availability. A distributed database involves a collection of number of logically related databases spread over a computer network. The software that manages distributed database is the distributed database management system, and it makes the distribution transparent to the users. The combination of DDB and DDBMS is referred to as distributed database system (DDBS). Internally, DBMSs perform several functions in order to manage and manipulate the data properly such as transaction management, concurrency control, recovery, etc. Transaction management tackles the issues of maintaining the database in a stable state at all times, even though there is simultaneous access or execution [1]. The concurrency control guarantees the consistency and integrity of the databases in the concurrent execution environment. DBMSs support database sharing among various transactions. The simultaneous execution of transactions if not properly managed may lead to conflict [2]. Therefore, there is need to manage the concurrent execution of the transactions such that the consistency and integrity of the database systems can be ensured. Concurrency control is the process by which DBMS can maintain the consistency and integrity of the databases, even in case where several transactions are executed simultaneously. Distributed concurrency control provides a mechanism to synchronize distributed transactions in such a way that the ACID properties are not violated by their interleaved execution. These transactions are performed in a distributed database system where relevant data is hosted by a group of linked data servers [3]. Therefore, not only local dependencies need to be taken into account, but also dependencies involving multiple data servers.

The rest of the work is organized as follows: Section II reviews related work; Section III explores distributed transactions, ACID properties and different states of a transaction. Section IV discusses distributed concurrency control, techniques used in distributed systems, and the paper is concluded in Section V.

II. Related Work

There are different researches about distributed transactions and concurrency controls in a distributed environment. Some of the researches are highlighted thus.

Distributed database systems have their data replicated over numerous locations. Unlike the centralized database system where a single copy of the data is stored, data may be distributed over a network using horizontal and vertical fragmentation similar to projection and selection operations in Structured Query Language (SQL). The same issue of access control and transaction management as simultaneous access and deadlock detection and resolution is shared by all types of databases. However, distributed databases, on the other hand, must cope with different problems. Access control and transaction management in Distributed Database Systems (DDBS) require various mechanisms and rules for controlling data retrieval and updating replicated and distributed databases as [4] posited. The study discusses transaction management in distributed database management system (DDBMS) and how this technique is implemented by Oracle. However, techniques other than two-phase commit were not explored. Several factors that affect concurrency in a distributed system were not also considered.

Distributed database is widely used in recent time. It is the arrangement of databases though logically interrelated, are spread across different physical locations. As data is dispersed across many sites, there is need for them to be synchronized [5]. The study explores various techniques used for the synchronization and concurrency control of distributed databases. The study also addresses different strategies for data synchronization, several clock synchronization algorithms, and explains the synchronization using audit log for homogeneous databases. Nevertheless, the use of efficient scheduler and other factors that facilitate transactions management were not explored.

The majority of organizations' applications rely on database systems. You can either implement a database system based on a centralized or distributed approach. In a distributed system, access control and concurrency becomes complex as compared to a centralized framework. A comparative analysis of different two-phase lock based concurrency control techniques have been done by [6]. The study discusses various lock based concurrency control techniques for distributed database management systems. It was however suggested that more researches concentrate on the most suitable and effective approach in maintaining concurrency in distributed database systems with high activity ratio.

Distributed concurrency control provides methods for synchronizing distributed transactions in such a way that the ACID properties are not violated by their interleaved execution. Distributed transactions are executed in a distributed environment where a series of linked data servers host related data [3]. The study explores diverse concurrency control techniques involved in ensuring that several transactions are executed simultaneously while maintaining the ACID properties of the transactions and serialization in the schedules. Distributed two-phase locking algorithm, and distributed timestamp concurrency control algorithm were also identified as concurrency control techniques in distributed systems. Factors that influence concurrency control in a distributed framework were not explored, however.

It is not easy to handle transactions in a distributed system as it uses heterogeneously networked systems to solve a single problem. As deadlines are imposed on the response time of the database system and transaction processing, the complexity of the distributed database increases [7]. The study addresses the management of transactions in a distributed real-time database. The set of databases logically correlated over heterogeneous networks where their transactions have explicit timing constraints, represented in the form of a deadline, is a distributed real-time database system. A deadline indicates that a transaction is expected to complete within or before a certain time. A model for transaction processing in a real-time database was proposed by the study. Workload arrival rate, concurrency protocol and resource time are handled by the proposed model. A slack time factor influences the deadline schedule. However, it was noted that further research is needed to design new algorithms, protocols, and transaction management techniques in a distributed real-time database. [8] states that, at each site, transactions and sub-transactions are scheduled to the CPU by the scheduler based on their priorities. It was reiterated that after processing all the operations of a transaction, the transaction enters in the validation phase. The study proposed a user control distributed database model which attempts to stimulate the overload transaction during run time. It was however suggested that scalability factor for a global implementation should be considered for further studies.

III. Distributed Transactions

Distributed transactions are executed in a distributed database framework where a set of connected data servers host related data. A distributed transaction comprises a set of sub-transactions, each of which is executed by a specific data server. These transactions are serialized to ensure effective execution. Having briefly described what a distributed transaction is, the question now is what is a transaction? The definition of a transaction and its properties are briefly discussed in section 3.1 and 3.2.

3.1 A Transaction

A Transaction consists of a series of read or write operation performed on a database. The read or write operation performs a specific unit of work. A transaction is a collection of read/write operations succeeding only if all contained operations succeed. However, a transaction depends on two outcomes, success or failure.

When a transaction starts execution, it could be terminated by two possibilities, which are Abort or Commit. A transactions is said to be aborted when it does not continue to a successful end or its not successfully completed whereas a transaction is committed when execution is successfully done and the changes stored accordingly. A database transaction must be atomic, consistent, isolated and durable. This is often referred to as the ACID Properties.

3.2 Properties of a Transaction

There is need to ensure data integrity in a concurrent environment. The ACID properties help to maintain the integrity of a database in terms of transactions. These properties describe the major guarantees of a database transaction. There are four properties of a transaction. These properties are represented by the acronym, ACID, which means Atomicity, Consistency, Isolation and Durability.

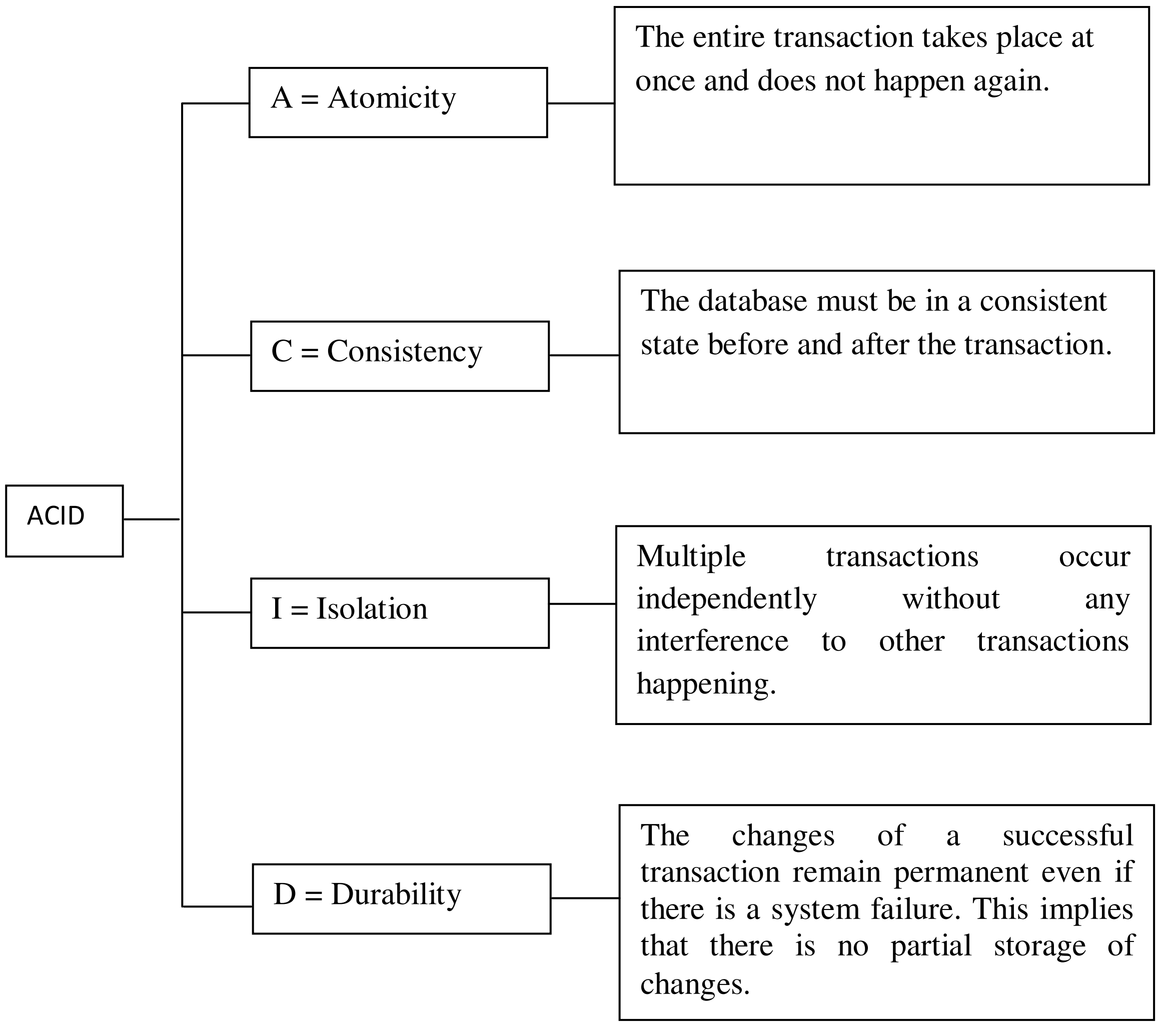
ACID
A = Atomicity
The entire transaction takes place at once and does not happen again.
C = Consistency
The database must be in a consistent state before and after the transaction.
I = Isolation
Multiple transactions occur independently without any interference to other transactions happening.
D = Durability
The changes of a successful transaction remain permanent even if there is a system failure. This implies that there is no partial storage of changes.

Figure 1: ACID Properties. Adapted from Transaction Management in Distributed Database Systems: the Case of Oracle's Two-Phase Commit (Alkhatib & Labban, 2020).

3.3 Different States of a Transaction

There are different states of a transaction, which include active state, partially committed state, committed state, failed and terminated states.

i. Active State: A transaction is said to be in the active state as soon as the execution begins. Read/write operations can be performed at this stage.

ii. Partially Committed State: At the end of a transaction, it is partially committed.

iii. Committed State: A transaction is in a committed state when the execution has been successfully completed, and changes stored or saved accordingly.

iv. Failed State: a transaction that is aborted while still in its active state is said to be a failed transaction.

v. Terminated State: A transaction is said to be terminated when certain transactions which are leaving the system cannot be continued or restarted.

IV. Distributed Concurrency Control

The task of managing concurrent access to a database in a distributed system is referred to as Distributed Concurrency control. It allows users to access a multi-programmed database while ensuring that transactions or processes are separated such that each user appears to be running or executing alone on a dedicated device or system.

Concurrency control involves the coordination among concurrent accesses to maintain consistency and integrity of the database. The key challenge in achieving this goal is to ensure that the changes or database updates performed by one transaction do not affect the updates and retrievals of another transaction. In case of concurrent execution of transactions, the consistency of the database can be ensured with the help of serializable schedules because the result obtained will be equivalent to one of the serial execution of the transactions. The use of only serial schedules restricts the degree of performance thereby impacting concurrency. Thus, to ensure that the schedules created by the simultaneous execution of transactions are serializable, concurrency control techniques must be applied as [9] posited.

4.1 Challenges Associated with Distributed Transactions and Concurrency Control

The challenges with concurrency control in a distributed system are highlighted below.

i. Data may be accessed by multiple users at a number of distant sites. This may cause inconsistent retrieval and update problems.

ii. Database is fragmented and/or replicated across multiple sites.

iii. Concurrency control techniques implemented at one location must ensure the consistency of the database at all other sites.

4.2 Benefits of Concurrency Control in Distributed Transactions

The benefits of concurrency control in distributed transactions are as follows:

i. Faster execution and response

ii. Improved performance

iii. It helps ensure serializability

iv. Reliability

4.3 Distributed Concurrency Control Techniques

There are various concurrency control mechanisms for managing concurrent access in a distributed system. In the distributed database system, concurrency control is used to resolve conflict with the concurrent access or update of data. The different techniques are discussed below.

4.3.1 Lock-Based Technique

The lock-based technique uses the concept of locking data items. A lock simply means a variable associated with a data item that determines whether read or write operation can be performed on that data item. Transactions indicate their intentions by requesting locks from the scheduler. The lock could either be a read or write lock. This technique helps to eliminate the concurrency problem in DBMS for simultaneous transactions by locking or isolating a particular transaction to a user. Transactions proceed only when the lock request is granted.

The Lock-based techniques are of two types:

i. Binary Lock
ii. Shared/Exclusive Lock

In binary lock, there two conceivable states namely locked and unlocked, which implies either a lock is acquired or not. A distinct lock is associated with each data item. Figure 2 shows that T1 is accessing the variable X thus, T2 should not be allowed to access X. If the lock is released by T1, then T2 can be allowed to access X.

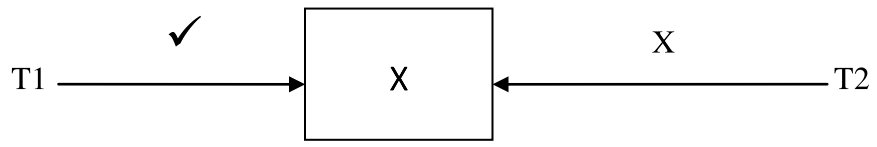

Figure 2: Binary Lock. Reprinted from A Review on Synchronization and Concurrency Control Techniques of Distributed Databases by [3]

However, for a shared lock, all transactions can acquire the lock on same data item X for reading purpose only. When a shared lock is acquired, another shared lock can also be obtained. However, update cannot be performed on any data item. It is also referred to as a read-only lock. Nevertheless, when an exclusive lock is acquired, no other lock can be acquired. This implies that access is denied to other transactions. The lock on a data item can only be released when the write operation is completed.

4.3.2 Timestamp-Based Technique

In this technique, the transactions are ordered based on some value referred to as timestamp. The timestamp can be either a value generated by a global incremental counter variable or current timestamp of the master clock which ensures the uniqueness of the timestamp. The transaction which is created earlier is assigned a lower timestamp as compared to the transaction which is created later. The transaction with lesser timestamp is older than the transaction with higher timestamp. Therefore, the ordering of transaction can be done based on the assigned timestamp and it is referred to as timestamp-based ordering technique which avoids the deadlock situation. This technique has the advantage of eliminating deadlocks as the transactions need not to wait.

Timestamp-based concurrency technique uses a transaction's timestamp to coordinate concurrent access to a data item to ensure serializability. A timestamp is a unique identifier assigned by the DBMS to a transaction. This represents the transaction's start time. These algorithms ensure that transactions commit in the order dictated by their timestamps. The transaction that was initiated earlier should commit before a younger transaction, since the transaction commenced before the other. This technique generates a serializable schedule such that the equivalent serial schedule is arranged in the order of the age of the participating transactions.

4.3.3 Optimistic Concurrency Control Technique

Optimistic concurrency control executes transactions simultaneously and determines if the transaction execution outcome at commit time was serializable. That is, before committing, the DBMS validates the transaction against all the transactions that have been committed since or are currently in the validation process. Validating every transaction for serializability may have negative impact on performance in systems with low conflict rates. In this case, the serializability test is deferred to just before committing. In this method, the life cycle of a transaction is divided into three stages, which include the execution, validation and commit phases. A transaction fetches data items into memory during execution and performs operations upon them. At the validation phase, transaction performs checks to ensure that serializability is achieved before the commit phase when committing the changes to the database is done. If a transaction can commit, the DBMS copies the local writes to the database and returns results to the client. Otherwise, the transaction is aborted and local copies of the data are lost. Each transaction acquires a unique transaction ID before starting and then adds it to the local timetable of the server with its commit timestamp. The DBMS copies each modified record into a private workspace during transaction execution. This enables transactions to continue without delay by checking for conflicts while executing. Nevertheless, if there is much conflict, there will be a substantial effect on performance.

4.3.4 Deterministic Concurrency Technique

Deterministic scheduling supports simpler replication strategies. Here, all clients send their queries to a distributed coordination layer comprised of sequencers that order the transactions and assign them a unique transaction id. At the end, the sequencers batch all of the transactions that they collected and forward them to the servers that manage the partitions that contain the records that the transaction wants to access. At each server, another component called the scheduler acquires record-level locks from each sequencer in a predetermined order. That is, each scheduler processes an entire batch of transactions from the same sequencer in the transaction order predetermined by that sequencer before moving to the next batch from another sequencer. If the transaction cannot acquire a lock, then the DBMS queues it for that lock and the scheduler continues processing. Execution proceeds in phases. First, the read/write set of the transaction is analyzed to determine a set of participating servers; that is, all servers that read or update records and all the active servers that perform updates. Thereafter, the system performs all of the local reads. If data from these local reads are needed during transaction execution at other servers, then the system forwards them to the servers responsible for the respective transactions. At this point, non-active servers that perform no updates have completed executing and can release their data locks. Once an active server receives messages from all participants it expects to receive data from, it applies the writes to its local partition. At this stage, active servers deterministically decide to commit or abort the transaction, and the locks are released. The servers send their responses to the sequencer, which sends an acknowledgement to the client once all responses have arrived.

V. Conclusion

In this study, a fascinating, but succinct discussion has been provided on distributed transactions and distributed concurrency control. Distributed concurrency control techniques ensure that multiple transactions are executed simultaneously while maintaining the data integrity. Managing multiple transactions in a distributed system is complex. Thus, without efficient concurrency control mechanism, it is not practicable to maintain the integrity of the database. The various mechanisms for concurrency control in a distributed system have been explored in this study. In terms of performance, the optimistic concurrency control technique is suitable where there is low traffic or data contention. When there is less conflict, without delaying for transactions' locks to be released, other transactions can complete, achieving higher throughput. Nevertheless, if there is much conflict, there will be a substantial effect on performance. Thus, other techniques perform better under this condition. It is however found that the method used for serializing or prioritizing transactions also affects performance. While concurrency control techniques regulate concurrent data access, synchronization ensures the consistency of data across different sites. Nevertheless, the inability to effectively manage distributed transactions can lead to deadlock or slow performance.

References

[1]. Ahmad, W., Abdul, W. M., Nadeem, M., Zeeshan, B., Mostapha, K., & Asadullah, S. (2015). Transaction management techniques and practices in current cloud computing environments: A survey. *International Journal of Database Management Systems, 7*(1), pp. 42 – 46.

[2]. Tabrez, Q. (2013). An efficient approach for concurrency control in distributed database system. *Indian Streams Research Journal, 3*(9), pp. 1 – 4.

[3]. Dilium, B. (2018). Distributed transactions and concurrency control. Retrieved July 2, 2020 from www.slideshare.net

[4]. Alkhatib, G., &Labban, R. S. (2020). Transaction management in distributed database systems: the case of Oracle's two-phase commit. *Journal of Information Systems Education, 13*(2), pp. 95 – 102.

[5]. Khan, S. R., Malik, M. S. A., Ashraf, M. W., Ullah, A. S., Asghar, I., & Razzzaq, N. (2019). A review on synchronization and concurrency control techniques of distributed databases. *International Journal of Computer Science and Network Security, 19*(20), pp. 187 – 192.

[6]. Gupta, M. K., Arora, R. K., & Bhati, B. S. (2018). Study of concurrency control techniques in distributed DBMS. *International Journal of Machine Learning and Networked Collaborative Engineering, 2*(4), pp. 180 – 187.

[7]. Abdalhalim, F. A., & Salih, A. K. (2015). A survey on transactions management in distributed real time database systems. *International Journal of Advanced Computer Technology, 4*(6), pp. 100 – 102.

[8]. Ranjana, J., Kamaljit, I. K., & Sanjay, G. (2015). Concurrency control model for distributed database. *International Journal of Engineering and Technology, 2*(1), pp. 55 – 58.

[9]. Sah, M. K., Kumar, V., & Tiwari, A. (2014). Security and concurrency control in distributed database system. *International Journal of Scientific Research and Management, 2*(12), pp. 1839 – 1845.