Volume 3, Issue 7, July 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Concurrency Control in Distributed Database System

Mandeep Kaur[*]
CSE (M.Tech)
Sant Baba Bhag Singh Institute of Engg. & Technology,
Jalandhar (Punjab), India

Harpreet Kaur
CSE (M.Tech)
Lovely Professional University,
Phagwara(Punjab), India

Abstract: This paper reviews the coverage of concurrency control in Distributed Network. A distributed network becomes more popular, the need for improvement in distributed database management systems becomes even more important. The main challenges are identified as-: (1)Preserving the ACID property atomicity, consistency, isolation and durability property when concurrent transactions perform read and write operation; (2) provides recovery method when distributed data is fail; (3)whatever method that is chosen they must provide feasible solutions with respect to performance.

Keywords: Distributed Database, Distributed Design, Fragmentation, Replication, Allocation, Concurrency control, Transaction

I. Introduction

In today's world of universal dependence on information systems, with the rising need for secure, reliable and accessible information in today's business environment, the need for distributed databases and client/ server applications is also increasing. A **distributed database** is a single logical database that is spread physically across computers in multiple locations that are connected by data communication links. **Distributed database** is a kind of virtual database whose component parts are physically stored in a number of distinct real databases at a number of distinct locations. The users at any location can access data at anywhere in the network as if the data were all stored at the user's own location. A distributed database management system is the software that manages the Distributed Databases, and provides an access mechanism that makes this distribution transparent to the user. The objective of a distributed database management system (DDBMS) is to control the management of a distributed database (DDB) in such a way that it appears to the user as a centralized database. This image of centralized environment can be accomplished with the support of various kinds of transparencies such as: Location Transparency, Performance Transparency, Copy Transparency, Transaction Transparency, Transaction Transparency, Fragment Transparency, Schema Change Transparency, and Local DBMS Transparency. Concurrency control is also an important issue in database systems. Concurrency control is the process of coordinating concurrent accesses to a database in a multi-user database management system (DBMS). There exist a number of methods that provide concurrency control. Some of the methods are: Two phase locking, Time stamping, Multi-version timestamp etc.

II. Motivation

There are various business conditions that encourage the use of distributed databases:

• **Data communications costs and reliability:** If the data is geographically distributed and the applications are related to these data, it may be much more economical, in terms of communication costs, to partition the application and do the processing at each site. On the other hand, the cost of having smaller computing powers at each site is much more less than the cost of having an equivalent power of a single mainframe

• **Database recovery:** Replicating data on separate computers is one strategy for ensuring that a damaged database can be quickly recovered and users can have access to data while the primary site is being restored. Replicating data across multiple computer sites is one natural form of a distributed database.

• **Data sharing:** Even moderately complex business decisions require sharing data across business units, so it must be convenient to consolidate data across local databases on demand.

• **Distribution and autonomy of business units:** Divisions, departments, and facilities in modern organizations are often geographically distributed, often across national boundaries. Often each unit has the authority to create its own information systems, and often these units want local data over which they can have control. Business mergers and acquisitions often create this environment.

•**Improved Performance**: Data retrieved by a transaction may be stored at a number of sites, making it possible to execute the transaction in parallel. Besides, using several resources in parallel can significantly improve performance.

•**Increased reliability and availability*:*** When a centralized system fails, the database is unavailable to all users. In contrast to centralized systems, distributed system will continue to function at some reduced level, however, even when a component fails.

•**Faster response:** Depending on the way data are distributed, most requests for data by users at a particular site can be satisfied by data stored at that site. This speeds up query processing since communication and central computer delays are minimized. It may also be possible to split complex queries into sub-queries that can be processed in parallel at several sites, providing even faster response [6].

III. DISTRIBUTED DATABASE DESIGN

Distributed Database Systems are needed for the applications where data and its accesses are inherently distributed and to increase the availability during failures. The prime examples are the systems for international air-line reservations, financial institutions, and automated manufacturing.

The methodology for designing Distributed Systems is same as that used for the Centralized databases. However, some additional factors have been considered for a Distributed Database:

A. Data Fragmentation:

In Distributed Databases, we need to define the logical unit of Database Distribution and allocation. The database may be broken up into logical units called fragments which will be stored at different sites. The simplest logical units are the tables themselves. Three Types of Data Fragmentation are:

- **Horizontal fragmentation:** A horizontal fragment of a table is a subset of rows in it. So horizontal fragmentation divides a table 'horizontally' by selecting the relevant rows and these fragments can be assigned to different sides in the distributed system [1].

Example: Consider a relation PROJ (PNo, Pname, Bugdet, Location)

PROJ

PNo	Pname	Budget	Location
P1	Banking	250000	Mumbai
P2	CAD/CAM	500000	Hyderabad
P3	Insurance	400000	Pune

PNo	Pname	Budget	Location
P2	CAD/CAM	500000	Hyderabad
P3	Insurance	400000	Pune

PROJ2

PROJ1

PNo	Pname	Budget	Location
P1	Banking	250000	Mumbai

Conditions-: PROJ1: Fragment with budget less than RS. 300000/-
PROJ2: Fragment with budget greater than or equal to Rs. 300000/-

- **Vertical fragmentation:** A vertical fragment of a table keeps only certain attributes of it. It divides a table vertically by columns. It is necessary to include the primary key of the table in each vertical fragment so that the full table can bere constructed if needed.

Example: Consider a Relation PROJ (PNo, Pname, Budget, Location)

PROJ

PNo	Pname	Budget	Location
P1	Banking	250000	Mumbai
P2	CAD/CAM	500000	Hyderabad
P3	Insurance	400000	Pune

PROJ2

PROJ1

PNo	Budget
P1	250000
P2	500000
P3	400000

PNo	Pname	Location
P1	Banking	Mumbai
P2	CAD/CAM	Hyderabad
P3	Insurance	Pune

- **Hybrid fragmentation:** Hybrid Fragmentation comprises the combination of characteristics of both Horizontal and Vertical Fragmentation. Each fragment can be specified by a SELECT-PROJECT combination of operations. In this case the original table can be reconstructed be applying union and natural join operations in the appropriate order.

B. Data Replication:

A popular option for data distribution as well as for fault tolerance of a database is to store a separate copy of the database at each of two or more sites.A copy of each fragment can be maintained at several sites. Data replication is the design process of deciding which fragments will be replicated [1].

IV. Fundamentals of Transaction Management and Concurrency Control

Transaction: A transaction consists of a series of operations performed on a database. The important issue in transaction management is that if a database was in a consistent state prior to the initiation of a transaction, then the database should return to a consistent state after the transaction is completed.

Properties of Transaction: A Transaction has four properties that lead to the consistency and reliability of a distributed database. These are Atomicity, Consistency, Isolation, and Durability.

- **Atomicity:** This refers to the fact that a transaction is treated as a unit of operation. It dictates that either all the actions related to a transaction are completed or none of them is carried out.
- **Consistency:** The consistency of a transaction is its correctness. In other words, a transaction is a correct program that maps one consistent database state into another.
- **Isolation:** According to this property, each transaction should see a consistent database at all times. Consequently, no other transaction can read or modify data that is being modified by another transaction.
- **Durability:** This property ensures that once a transaction commits, its results are permanent and cannot be erased from the database. This means that whatever happens after the COMMIT of a transaction, whether it is a system crash or aborts of other transactions, the results already committed are not modified or undone.

Concurrency Control: In distributed database systems, database is typically used by many users. These systems usually allow multiple transactions to run concurrently i.e. at the same time. Concurrency control is the activity of coordinating concurrent accesses to a database in a multiuser database management system (DBMS). Concurrency control permits users to access a database in a multi-programmed fashion while preserving the illusion that each user is executing alone on a dedicated system. The main technical difficulty in attaining this goal is to prevent database updates performed by one user from interfering with database retrievals and updates performed by another.When the transactions are updating data concurrently, it may lead to several problems with the consistency of the data.

V. Distributed Concurrency Control Algorithms:

In this paper, we consider some of the distributed concurrency control algorithms. We summarize the salient aspects of these four algorithms in this section. In order to do this, we must first explain the structure that we have assumed for distributed transactions. Before discussing the algorithms, we need to get an idea about the distributed transactions.

Distributed Transaction: A distributed transaction is a transaction that runs in multiple processes, usually on several machines. Each process works for the transaction. Distributed transaction processing systems are designed to facilitate transactions that span heterogeneous, transaction-aware resource managers in a distributed environment. The execution of a distributed transaction requires coordination between a global transaction management system and all the local resource managers of all the involved systems. The resource manager and transaction processing monitor are the two primary elements of any distributed transactional system.

Distributed transactions, like local transactions, must observe the ACID properties. However, maintenance of these properties is very complicated for distributed transactions because a failure can occur in any process. If such a failure occurs, each process must undo any work that has already been done on behalf of the transaction.

A distributed transaction processing system maintains the ACID properties in distributed transactions by using two features:

- Recoverable processes. Recoverable processes log their actions and therefore can restore earlier states if a failure occurs.
- A commit protocol. A commit protocol allows multiple processes to coordinate the committing or aborting of a transaction. The most common commit protocol is the two-phase commit protocol.

- **Distributed Two-Phase Locking (2PL):**

In order to ensure serializability of parallel executed transactions elaborated different methods of concurrency control. One of these methods is locking method. There are different forms of locking method. Two phase locking protocol is one of the basic concurrency control protocols in distributed database systems. The main approach of this protocol is “read any, write all”. Transactions set read locks on items that they read, and they convert their read locks to write locks on items that need to be updated. To read an item, it suffices to set a read lock on any copy of the item, so the local copy is locked; to update an item, write locks are required on all copies. Write locks are obtained as the transaction executes, with the transaction blocking on a write request until all of the copies of the item to be updated have been successfully locked. All locks are held until the transaction has successfully committed or aborted [2].

The 2PL Protocol oversees locks by determining when transactions can acquire and release locks. The 2PL protocol forces each transaction to make a lock or unlock request in two steps:

- Growing Phase: A transaction may obtain locks but may not release any locks.
- Shrinking Phase: A transaction may release locks but not obtain any new lock.

The transaction first enters into the Growing Phase, makes requests for required locks, then gets into the Shrinking phase where it releases all locks and cannot make any more requests. Transactions in 2PL Protocol should get all needed locks before getting into the unlock phase. While the 2PL protocol guarantees serializability, it does not ensure that deadlocks do not happen. So deadlock is a possibility in this algorithm, Local deadlocks are checked for any time a transaction blocks, and are resolved when necessary by restarting the transaction with the most recent initial startup time among those involved in the deadlock cycle. Global deadlock detection is handled by a "Snoop" process, which periodically requests waits-for information from all sites and then checks for and resolves any global deadlocks.

- **Wound-Wait (WW):**

The second algorithm is the distributed wound-wait locking algorithm. It follows the same approach as the 2 PL protocol. The difference lies in the fact that it differs from 2PL in its handling of the deadlock problem: unlike 2PL protocol, rather than maintaining waits-for information and then checking for local and global deadlocks, deadlocks are prevented via the use of timestamps in this algorithm. Each transaction is numbered according to its initial startup time, and younger transactions are prevented from making older ones wait. If an older transaction requests a lock, and if the request would lead to the older transaction waiting for a younger transaction, the younger transaction is "wounded" – it is restarted unless it is already in the second phase of its commit protocol. Younger transactions can wait for older transactions so that the possibility of deadlocks is eliminated [2].

Example-: Wound-Wait algorithm

	T1 is allowed to
t(T1) > t(T2) $\longrightarrow$	**Wait**
t(T1) < t(T2) $\longrightarrow$	**Abort and rolled back**

t(T1) > t(T2) -: If requesting transaction [**t(T1)**] is younger than the transaction [**t(T2)**] that has holds lock on requested data item then requesting transaction [**t(T1)**] has to wait.
t(T1) < t(T2) -: If requesting transaction [**t(T1)**] is older than the transaction [**t(T2)**] that has holds lock on requested data item then requesting transaction [**t(T1)**] has to abort or rollback.

- **Basic Timestamp Ordering (BTO):**

A timestamp is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the transaction start time. The third algorithm is the basic timestamp ordering algorithm. The idea for this scheme is to order the transactions based on their timestamps. A schedule in which the transactions participate is then serializable, and the equivalent serial schedule has the transactions in order of their timestamp values. This is called timestamp ordering (TO).

Like wound-wait, it employs transaction startup timestamps, but it uses them differently. BTO associates timestamps with all recently accessed data items and requires that conflicting data accesses by transactions be performed in timestamp order instead of using locking approach. Transactions that attempt to perform out-of-order accesses are restarted. When a read request is received for an item, it is permitted if the timestamp of the requester exceeds the item's write timestamp. When a write request is received, it is permitted if the requester's timestamp exceeds the read timestamp of the item; in the event that the timestamp of the requester is less than the write timestamp of the item, the update is simply ignored [2]. For replicated data, the "read any, write all" approach is used, so a read request may be sent to any copy while a write request must be sent to all copies. Integration of the algorithm with two phase commit is accomplished as follows: Writers keep their updates in a private workspace until commit time.

- **Distributed Optimistic(OPT):**

The fourth algorithm is the distributed, timestamp-based, optimistic concurrency control algorithm. which operates by exchanging certification information during the commit protocol. For each data item, a read timestamp and a write timestamp are maintained. Transactions may read and update data items freely, storing any updates into a local workspace until commit time. For each read, the transaction must remember the version identifier (i.e., write timestamp) associated with the item when it was read. Then, when all of the transaction's cohorts have completed their work, and have reported back to the master, the transaction is assigned a globally unique timestamp. This time stamp is sent to each cohort in the "prepare to commit" message ,and it is used to locally certify all of its reads and writes as follows [2]:

A read request is certified if-:

(i) The version that was read is still the current version of the item, and

(ii) No write with a newer timestamp has already been locally certified.

A write request is certified if-:

(i) No later reads have been certified and subsequently committed, and

(ii) No later reads have been locally certified already [2].

VI. Conclusion

In this paper we have discuss about the distributed database system that is considered to be more reliable than centralized database system. We also describe the concurrency control algorithms-: distributed 2PL, wound-wait, basic timestamp ordering and distributed optimistic algorithm.It is really important for database to have the ACID properties to perform.

References

[1] Gupta V.K., Sheetlani Jitendra, Gupta Dhiraj and Shukla Brahma Datta, *Concurrency control and Security issues in Distributed database system*, Vol. 1(2),70-73, August (2012)

[2] Arun Kumar Yadav& Ajay Agarwal, *An Approach for Concurrency Control in Distributed Database System*, Vol. 1, No. 1, pp. 137-141**,** January-June (2010)

[3] Navathe Elmasri, Database Concepts, *Pearson Education*, V edition (2008)

[4] Fundamentals of DBMS, Lakhanpal Publisher, III edition (2008)

[5] Swati Gupta, Kuntal Saroha, Bhawna, *Fundamental Research of Distributed Database*, IJCSMS International Journal of Computer Science and Management Studies, Vol. 11, Issue 02, Aug 2011

[6] Distributed Database: http://wps.pearsoned.co.uk/wps/media/objects/10977/11240737/Web%20chapters/Chapter%2012_WEB.pdf