Concurrency Control in Distributed Database System

1.Anshu Singla ,2.Anjani Kumar Singha, 3.Sudhir Kumar Gupta

1.Department of Computer Science,Jamia Millia Islamia, New Delhi

Email id::anshu008in@gmail.com

2.Department of Computer Science and Engineering, Gurukula Kangri Vishwavidyalaya Haridwar, Uttarakhand,Email:Anjani348@gmail.com

3.Guru jambheshwar university of science and technology,email: sudhirvns0542@gmail.com .

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: the process of restoring data that has been lost, accidentally deleted, corrupted or made inaccessible for any reason. In enterprise information technology (IT), data recovery typically refers to the restoration of data to a desktop, laptop, server, or external storage system from a backup**

• **Data sharing: the practice of making data used for scholarly research available to other investigators. Replication has a long history in science. The motto of The Royal Society is 'Nullius in verba', translated "Take no man's word for it."**

• **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**: Performance improvement, by its nature, is iterative. For this reason, removing the first bottleneck might not lead to performance improvement immediately, because another bottleneck might be revealed. Also, in some cases, if serialization points move to a more inefficient sharing mechanism, then performance could degrade. With experience, and by following a rigorous method of bottleneck elimination, applications can be debugged and made scalable

•**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: well-suited for loosely defined data structures that may evolve over time.** In-memory database management system (IMDBMS) - provides faster response times and better performance.

1...DISTRIBUTED DATABASE DESIGN:Distributed databases can be homogenous or heterogeneous. In a homeogenous distributed database system, all the physical locations have the same underlying hardware and run the same operating systems and database applications. In a heterogeneous distributed database, the hardware, operating systems or database applications may be different at each of the locations.

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: refers to the division of a relation into subsets (fragments) of tuples (rows).**

Example:

Say we have this relation

customer_id	Name	Area	Payment Type	Sex
1	Bob	London	Credit card	Male
2	Mike	Manchester	Cash	Male
3	Ruby	London	Cash	Female

Horizontal Fragmentation are subsets of tuples (rows)

Fragment 1

customer_id	Name	Area	Payment Type	Sex
1	Bob	London	Credit card	Male
2	Mike	Manchester	Cash	Male

Fragment 2

customer_id	Name	Area	Payment Type	Sex
3	Ruby	London	Cash	Female

- **Vertical fragmentation: Vertical Fragmentation of a relation R produces fragments R1, ..,Rn, each of.**

Example: Fragment 1

```
customer_id | Name | Area        | Sex
      1     | Bob  | London      | Male
      2     | Mike | Manchester  | Male
      3     | Ruby | London     Female
```

Fragment 2

```
customer_id | Payment Type
      1     | Credit card
      2     | Cash
      3     | Cash
```

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:**

the frequent electronic copying **data** from a **database** in one computer or server to a **database** in another so that all users share the same level of information. The result is a distributed **database** in which users can access **data** relevant to their tasks without interfering with the work of others

III. Fundamentals of Transaction Management and Concurrency Control

Transaction: is a sequence of operations performed as a single logical unit of work. A logical unit of work must exhibit four properties, called the atomicity, consistency, isolation, and durability (ACID) properties, to qualify as a transaction.

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: An atomic transaction is an indivisible and irreducible series of database operations such that either all occur, or nothing occurs.**
- **Consistency: database systems refers to the requirement that any given database transaction must change affected data only in allowed ways. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof**
- **Isolation: in the context of databases, specifies when and how the changes implemented in an operation become visible to other parallel operations. Transaction isolation is an important part of any transactional system. It deals with consistency and completeness of data retrieved by queries unaffecting a user data by other user actions. A database acquires locks on data to maintain a high level of isolation**
- **Durability: databases is the property that ensures transactions are saved permanently and do not accidentally disappear or get erased, even during a database crash. This is usually achieved by saving all transactions to a non-volatile storage medium**

Concurrency Control:A database management systems (DBMS) concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system.

IV. 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):**

Two-phase locking (**2PL**) is a concurrency control method that guarantees serializability. It is also the name of the resulting set of database transaction schedules (histories). The protocol utilizes locks, applied by a transaction to data, which may block (interpreted as signals to stop) other transactions from accessing the same data during the transaction's life

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(Expanding phase): locks are acquired and no locks are released.
- Shrinking Phase: locks are released and no locks are acquired..

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 It is a preemptive technique for deadlock prevention. It is a counterpart to the wait-die scheme. When Transaction Ti requests a data item currently held by Tj, Ti is allowed to wait only if it has a timestamp larger than that of Tj, otherwise Tj is rolled back (Tj is wounded by Ti).
In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur

Example-: Wound-Wait algorithm

	T1 is allowed to
t(T1) > t(T2)	**Wait**
t(T1) < t(T2)	**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.
Suppose that Transactions T22, T23, T24 have time-stamps 5, 10 and 15 respectively . If T22requests a data item held by T23, then data item will be preempted from T23 and T23 will be rolled back. If T24 requests a data item held by T23, then T24 will wait.

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

We assume that the Transaction Manager (TM) attaches an appropriate timestamp to all read and write operations. In the BTO, the scheduler at each Data Manager (DM), keeps track of the largest timestamp of any read and write operation processed thus far for each data object. These timestamps may be denoted by *R-ts(object)* and *W-ts(object)*, respectively. Let us also make the following notations:
read(x,TS) --> Read request with timestamp *TS* on a data object *x*.
write(x,v,TS) --> Write request with timestamp *TS* on a data object *x*. *v* is the value to be assigned to *x*.
A read request is handled in the following manner:
If *TS* < *W-ts(x)* **then**

reject read request and **abort** corresponding transaction
else
execute transaction
Set *R-ts(x)* to *max{R-ts(x), TS}*

A write request is handled in the following manner:
If *TS* < *R-ts(x)* or *TS* < *W-ts(x)* **then**

reject write request
else
execute transaction
Set *W-ts(x)* to *TS*.

If a transaction is aborted, it is restarted with a new timestamp. This can result in a cyclic restart where a transaction can repeatedly restart and abort without ever completing. Another disadvantage is that it has storage overhead for maintaining timestamps (two timestamps must be kept for every data object)

- **Distributed Optimistic(OPT):The fourth algorithm is Locking bad because:**
    - overhead not present in sequential case. Even read-only transactions need to use locks.
    - lack of general-purpose deadlock free locking protocols.
    - large parts of DB on secondary storage - concurrency significantly lowered when it is necessary to leave congested node while accessing disk
    - to allow transaction to abort, need to hold on to locks until EOT
    - locking may only be necessary in worst case
- optimistic: reading a value/pointer can never cause loss of integrity; reads are completely unrestricted, although returning a value from a query is considered equivalent to a write.
- writes are severly restricted. Have read phase, validation phase, then write phase. During read, all writes are on local copies. Validation ensures consistency across active transactions. Write phase is when local copies get written to global DB.
- concurrency control procedures maintain sets of objects accessed by transaction. maintain read and write set for transaction.
- serial equivalence: if have individual transactions that are executed concurrently, have serial equivalence if there exists some serial sequence of the transactions that produce the final data structure. serial equivalence is sufficient (but not necessary) for integrity of DB.
- validation of serial equivalence: assign each transaction a transaction number (TN), and explicitly force serializability. Need one of these three conditions (for i
- Ti completes write phase before Tj starts read phase
- write set of Ti does not intersect read set of Tj, and Ti completes its write phase before Tj starts its write phase
- write set of Ti does not intersect read set *or* write set of Tj, and Ti completes its read phase before Tj completes its read phase. (1) says Ti completes before Tj starts. (2) states writes of Ti don't affect read phase of Tj, and Ti finishes writing before Tj starts writing, hence doesn't overwrite Tj. (3) similar to (2), but does not require that Ti finish writing before Tj starts writing.

  Assign transactions numbers when transaction enters validation, to prevent fast transactions from blocking behind slow but earlier starting transactions.

- Serial validation: assign transaction nmber, validate, and do write phase all in critical section. Fine if one CPU and write phase can take place in main memory. Otherwise too inefficient - not enough concurrency. If multiple CPUs, be more clever about critical sections.
- Parallel validation: maintain set of active transactions (in read but not yet completed write phase); do validation against all transactions in active set. To prevent an aborted transaction from causing further aborts, do multistage validation.
- analysis for large B-trees shows that optimistic concurrency control is a win - very rare that one insertion would cause another concurrent insertion to restart. (Assumes random access to B-tree.)

V. 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: ttp://wps.pearsoned.co.uk/wps/media/objects/10977/11240737/Web%20chapters/Chapter%2012_WEB.phdf