International Journal of Modern Computation, Information and Communication Technology 2019;2(2):48-52.
ISSN: 2581-5954
http://ijmcict.gjpublications.com

Review Article

Challenges of Concurrency Control in Object Oriented Distributed Database Systems

Muhammad Haroon

University of Gujrat Lahore Sub Campus, Lahore, Pakistan.

*Corresponding author's e-mail: haroon.capricorn@gmail.com

Abstract

Concurrent data in any system defines the correctness and credibility of the system. It becomes trickier and challenging in case of object oriented distributed database systems. Concurrency control is the area in which we try to meet the maximum concurrency level in the system. The present review paper is focused on the challenges one can face regarding concurrency control in object oriented distributed database systems.

Keywords: Concurrency; Concurrency control; Distributed database; Object oriented distributed database.

Introduction

One of the key properties of ACID properties of database transaction is consistency that must be preserved during all database transactions. The data manipulated by these transactions must be concurrent. When a system works, more than one transaction try to manipulate the same data that may cause ill manipulation of data. To overcome this problem, transactions are made serialized that in turn gives the concept of concurrency control [1]. The concurrency control is a methodology that keeps data concurrent even when more than one transaction tries to read or write same data cluster [3]. In the case of object oriented distributed database system, it becomes more challenging and requires more attention to work with it because the architecture of distributed database system is very different from the regular single site database. There are several techniques to handle such type of situations.

Object oriented distributed database

Distributed database is same like a regular centralized database but it is physically spread across multiple geographical sites and is connected through wired or wireless network. It is done to boost up the transactions for local users for that particular site [3].

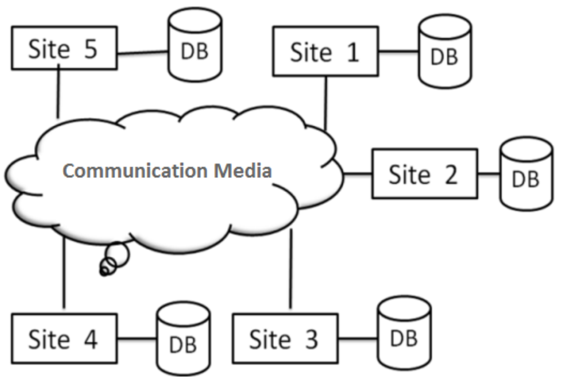

Fig. 1. Architecture of distributed database

Fig. 1 demonstrate graphically the concept of distributed database system in which a single database is spread around five sites and all fragments of database are connected by communication media with each other.

Concurrency

Concurrency is actually one of the main properties of relational database system that allows multiple users to affect multiple transactions at a time. This differentiates the database from other storage modes like traditional file systems, spreadsheets etc [4].

Consider one banking transaction demonstrated graphically in fig. 2. Suppose there's a bank account with amount Rs.5000. Two transactions T1 and T2 try to work on data

Received: 30.06.2019; Received after Revision: 13.07.2019; Accepted: 14.07.2019

of bank account. Transaction T1 adds Rs.1000 and on the same time, transaction T2 withdraws Rs.6000. These both transactions affect the same data at the same time. In that case if transaction T2 executes before T1, an error containing message “Insufficient Funds” will appear but it will be withdrawn if T1 runs before T2. In this case, T1 must be executed first. This is done using serialization in concurrency control. Transactions are made serialized [6].

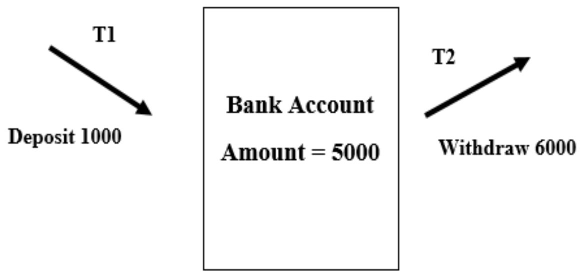

Fig. 2. Bank transactions

Transactions in distributed environment become a little bit complex. This can be termed as distributed transaction. In this case, transactions can be controlled by local transaction manager at each host computer. This transaction manager communicates with other host transaction to preserve concurrency control [3,4]. These distributed transaction managers are built to facilitate transactions to work in smooth manner by avoiding collision or damage of data [5].

How Concurrency can be controlled?

The correctness of data is one of the major concerns in any system that must be fulfilled in any case. It is said that a system’s worth is of its data, not the system itself. Systems where more than one transaction executes with time overlap are most critical to be considered. After getting maturity in 1970s, researchers were started their research to control the concurrency because it was the one of the biggest concern of that time [6]. Some theories had also been developed from which the serializability theory became famous and workable. According to this theory, transactions are prioritized based upon their importance and effect on the system to retain data integrity in the system [7]. Serializability works on the principle of making schedule of running transactions to avoid transaction collision or data damage. In this way, concurrency is controlled.

5. Techniques of Concurrency Control

Serializability is the technique to stop a transaction temporarily while other transaction is accessing some data item. There are various methods to control the concurrency. Few are mentioned below:

I. Distributed Two-Phase Locking Protocol
    a. Multi version Two-Phase Locking
II. Timestamp-Based Protocols
    a. Multi version Timestamp Ordering
    b. Wait-Die & Wound-Wait Algorithms
III. Validation-Based Protocols

Distributed two-phase locking protocol

It is the most common protocol in relational databases. This works in two phases i.e. growing phase and shrinking phase as shown in fig. 3.

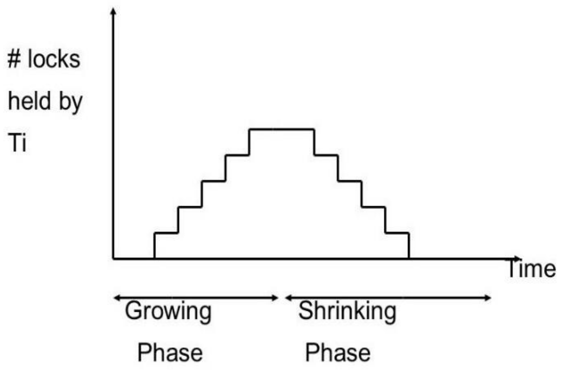

Fig. 3. Two phase locking protocol (2PL)

In first phase, locks are acquired only and cannot be released and are released in second phase and there is no new lock acquired in this second phase [7].

A transaction in distributed environment is granted a lock on a transaction if the requested lock is suitable with another lock already held by other transactions. More than one transaction can hold shared locks on a transaction but if there is any transaction that holds another lock on the item no other transaction may hold any lock on a transaction. If a lock cannot be acquired, the transaction in queue is waited till all unsuitable locks held by other transactions have been released [7].

Table 1. Advantages and disadvantages of two phase locking protocol

Advantages	Disadvantages
Transactions are executed with changed manner to ensure serializability and to avoid deadlock.	This protocol sometimes becomes overhead when a transaction traps in a deadline causing rolling back again and again.
It assures no conflict in running transactions.	It does not assure the occurrence of deadlocks and starvation.

Another variant of distributed two phase locking is multi version two phase locking.

Multi version two-phase locking

Multi version refers to the creation of a new version after every commit rather than to overwrite old values. This variant of multi version protocol maintains one or more than one old versions of items in the database to allow work to proceed using both the current version and older versions [8]. In ordinary two phase locking, all locks are hold till the end of the transaction. Write lock on a data saves a transaction from setting a read locks on data. When a transaction writes into data, it creates a new version of the data [8].

Table 2. Advantages and disadvantages of multi version two phase locking protocol

Advantages	Disadvantages
It is best for read only transactions.	Deadlock avoidance can never be neglected.
All read requests are always accepted.	-

Timestamp-Based Protocols

Timestamp based protocols work with an additional 'timestamp' parameter which is associated to every transaction. The prioritization of the transaction execution is based upon the timestamp. There are two timestamps i.e. read timestamp and write timestamp. This is suitable in the case described in fig.2. If the timestamp of transaction T1 is less than transaction T2, the request is granted. In this way the transactions are executed on the basis of timestamps [4][9].

Table 3. Advantages and disadvantages of timestamp based protocols

Advantages	Disadvantages
All updates are set into the database after the transaction rolls back.	When a transaction halts, it may be possible that it possesses the data held by other transaction. In this case, transaction must be rolled back [5].
Transactions sometimes produce conflict due to reading similar kind of data.	Due to the halt of a transaction, will be restarted with another timestamp.

This protocol guarantees the serializability between transactions performing read and writes operations. The rule based on algorithm can be written as:

```
Transaction T1 starts P(A) operation.
If (Write_TimeStamp > TimeStamp (T1))
        Then rollback T1
Else
        Execute P(A)
and set P_ TimeStamp(A) = MAX{ P_ TimeStamp (A), TimeStamp (T1)}.
```

Some of the variants of timestamp based protocols are defined in following.

Multi version Timestamp Ordering

As described earlier, multi version refers to the creation of a new version after every commit rather than to overwrite old values. Multi version timestamp ordering works on FCFS (First Come First Server) basis [9]. The transaction comes first will be treated first. Whenever a transaction is allowed to execute a write command, a new version of data item is generated. A read is always executed first always when it finds the suitable version to read based on the write of the various existing versions of item based on timestamp [9].

Table 4. Advantages and disadvantages of multi version timestamp ordering

Advantages	Disadvantages
In this approach, deadlock can't occur.	If there are conflicts in transactions, these are sorted by rollbacks [9].

Wait-Die & Wound-Wait Algorithms

Wait-Die: When a transaction T1 requests a data held by T2, then T1 is allowed to wait only if it has a timestamp less than T2, otherwise T1 is rolled back [4].

- *If TimeStamp(T1) < TimeStamp (T2)*
  T1 is allowed to wait until the data is available.
- *If TimeStamp (T1) > TimeStamp (T2)*
  T1 is restarted later with some delay but with the same timestamp.

Wound-Wait: When Transaction T1 requests a data held by T2 then T1 is allowed to wait only if it has a timestamp larger than that of T2, otherwise T2 is rolled back [5].

If TimeStamp (T1) < TimeStamp (T2)
T1 makes T2 to be rolled back
If TimeStamp (T1) > TimeStamp (T2)
T1 is forced to wait until the required data is available.

Validation-based protocols

Locking mechanisms that are deadlock free are the most favorable for us. A research conducted in [11] suggested that there must be another phase “validation phase” along with write phase and read phase [11]. In this new validation phase, it is assured that transactions are not violating the serializability pattern. Validation is achieved by giving each transaction a timestamp at the end of the read phase and synchronizing using timestamp ordering. This produces a term optimistic concurrency control [9].

Table 5. Advantages and disadvantages of validation based protocol

Advantages	Disadvantages
This can achieve a better level of concurrency with a very low conflict rate.	There is a chance of transaction starvation due to conflicting short transactions.

Issues with techniques of concurrency control

Some issues can arise with the techniques of concurrency control was discussed in section 5 of this paper. Some of the issues are described here.

Two phase locking

Two phase locking protocol works well for read-update applications. In this two phase locking approach, transactions are controlled by letting them wait at some points [12]. The major hurdle in this locking technique is deadlock occurrence which can be solved by using backup and these cannot work well with query intensive applications [12].

Timestamp ordering

The timestamp ordering protocol guarantees serializability of transactions as conflicting issues are fixed by timestamp order [3,12]. As we know, no any transaction waits for each other transaction; therefore this protocol ensures liberty from deadlocks. There are some chances of starvation of lengthy transactions because the sequence of conflicting short transactions may repeatedly have restarted [13]. There is a case when ordering is incorrect like transaction that started later than the current transaction has accessed the file and committed, in this case the current transaction is late and has to halt [13].

Multi version concurrency control

The concurrency control scheduler usually rejects a read transaction because the value it was required to read has already been overwritten but with multi version concurrency control, since these values are never overwritten because it produces a new version with every successful write [10].

There is a tradeoff between concurrency control and the memory. Multi version concurrency control increases the cost of storing multiple versions in storage media. This storage requirement can be managed by archiving old versions in backup [10].

Conclusions

After all discussion we made in this paper, it is finally concluded that the concurrency is one of the important factor that can’t be ignored or kept back sided. The credibility of the system is developed with the concurrency control in the system. Various methods to control the concurrency have been discussed here with advantages and disadvantages of each method. It has also been discussed that implementation of these methods becomes more challenging with object oriented distributed database systems. But still the under discussion methods are enough to meet a handsome level of concurrency.

Conflicts of interest

Authors declare no conflict of interest.

References

[1] Korth HF, Silberchatz A, Sudarshan S. Concurrency Control: Database System Concepts (4th Edition), Bell Laboratories, USA. 2001. pp. 591-617.

[2] Su C, Crooks N, Ding C, Alvisi L, Xie C. Bringing Modular Concurrency Control To The Next Level. In Proceedings Of The 2017 Acm International Conference On Management Of Data, Sigmod Conference 2017, Chicago, Il, USA, May 14-19, 2017;283-97.

[3] Zamanian E, Binnig C, Harris T, Kraska T. The End Of A Myth: Distributed Transactions Can Scale. Proceedings of the VLDB Endowment 2017;10(6):685-96.

[4] Curino C, Zhang Y, Jones EPC, Madden, S. Schism: A Workload-Driven Approach To Database Replication And Partitioning. Proceedings of the VLDB Endowment 2010;3:48-57.

[5] Zheng W, Tu S, Kohler E, Liskov B. Fast Databases With Fast Durability And Recovery Through Multicore Parallelism. In 11th Usenix Symposium On Operating Systems Design And Implementation, Osdi ’14, Broomfield, Co, USA, 2014;465–477.

[6] Jitendra S, Gupta VK. Concurrency Issues of Distributed Advance Transaction Process. Research Journal of Recent Sciences 2012;1:426-29.

[7] Tatarowicz A, Curino C, Jones EPC, Madden S. Lookup Tables: Fine-Grained Partitioning For Distributed Databases. In Ieee 28th International Conference On Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012:102–113.

[8] Christos H. Papadimitriou: A Theorem In Database Concurrency Control. Journal Of The. Association for Computing Machinery 1982;29:998-1006.

[9] Kimura, H. Foedus: Oltp Engine For A Thousand Cores And Nvram. In Proceedings Of The 2015 ACM Sigmod International Conference On Management Of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015:691-706.

[10] Kim K, Wang T, Johnson R, Pandis I. Ermia: Fast Memory-Optimized Database System for Heterogeneous Workloads. In Proceedings of The 2016 International Conference on Management of Data, Sigmod Conference 2016, San Francisco, CA, USA, June 26-July 01, 2016;675-87.

[11] Diaconu C, Freedman C, Ismert E, Larson, P, Mittal P, Stonecipher R, Verma N, Zwilling M. Hekaton: Sql Server’s Memory-Optimized Oltp Engine. In Proceedings Of The ACM Sigmod International Conference on Management of Data, Sigmod 2013, New York, NY, USA, June 22-27, 2013;1243–1254.

[12] Didona D, Diegues N, Kermarrec A, Guerraoui R, Neves R, Romano P. Proteustm: Abstraction Meets Performance in Transactional Memory. In Proceedings of The Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, Asplos ’16, Atlanta, GA, USA, April 2-6, 2016;757-71.

[13] Shang Z, Li F, Yu JX, Zhang Z, Cheng H. Graph Analytics Through Fine-Grained Parallelism. In Proceedings of The 2016 International Conference on Management Of Data, Sigmod Conference 2016, San Francisco, CA, USA, June 26-July 01, 2016;463-78.

****