EasyChair Preprint

№ 4211

Concurrency Control in Distributed Database

Fatima Bara, Chaimae Saadi and Habiba Chaoui

September 19, 2020

Concurrency Control in Distributed Database

Fatima BARA [1], Chaimae SAADI [2], Habiba CHAOUI [3]

Laboratory for Systems Analysis, Information Processing and Industrial Management

Systems Engineering Laboratory

fatima.bara97@gmail.com [1] , chaimaesaadi900@gmail.com [2], habiba.chaoui@uit.ac.ma [3]

ENSA Kenitra Ibn Tofail University [1] – EST Salé Mohamed V University [2] - ENSA Kenitra Ibn Tofail University [3]

Abstract **In a distributed database, uncontrolled concurrent execution of multiple transactions can lead to inconsistencies and can affect data integrity. So, there is requirement of controlling the concurrent execution of the transactions so that the consistency and integrity of the database systems can be ensured. The study presents the new breed of mechanism known as the SC-2PL concurrency control mechanism, which is a mixture of Strict-2PL and Conservative-2PL concurrency control mechanisms. Our proposal improves the performance of the distributed database through the elimination of blockages, cascading aborts and dirty reads and write.**

Keywords distributed database, concurrency control, Strict-2PL, Conservative-2PL, SC-2PL

I. INTRODUCTION

With the fast development of the Internet, the traditional stand-alone database cannot satisfy the needs of massive storage of logs and concurrent access by a large number of users, which is why the distributed database came into being. but no data storage solution is without its drawbacks. there are several problems in distributed databases and among these problems there is uncontrolled concurrent that affect data integrity.

In a distributed database system, A major problem in these systems is concurrency control to general data parallel executed transactions [1]. several transactions access the same data element at the same time. In addition, the simultaneous operation of the database must be controlled to guarantee consistency and integrity of database, and to avoid faults such as data loss, data inconsistency, and dirty reads or write.
Many methods of concurrency control have already been developed, but they pose problems of delay, performance, deadlock, rollback...

The algorithm we propose aims to increase concurrency degree of concurrent transactions. and to find a good scheduling strategy that can prevent blocking and rollback.

The rest of this paper organized as follows: Sect. 2 describes the previous related work. Section 3 introduces the proposed algorithm. Finally, Sect. 4 concludes the paper.

II. RELATED WORK

1. 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[2]. These distributed databases can be stored either on a network server, on corporate intranets and extranets, or, more newly and more commonly, on decentralized stand-alone computers located on the Internet. There are two types of distributed database management systems (DDBMS), homogeneous and heterogeneous.

In a homogeneous DDBMS: all the computer sites have the same database product installed and the operating system is also the same[3].

In heterogeneous DDBMS: some sites have different database products installed and the underlying data model is also different[3].

2. Concurrency Control

There have been recent improvements in these concurrency control algorithms in order to resolve some of the concurrency issues and to achieve better distributed database performance. This study can be summarized in a few research studies such as:

In 2018, the researchers proposed an improvement of 2PL to achieve deadlock free cell locking (DFCL). In[4] this protocol provides less percentage of conflict, this reduces the need to transaction abortion

in addition to reducing the average number of rollbacks. DFCL is proposed to eliminate the deadlock of the locking algorithms. Also, DFCL improves database performance via increasing the number of commits, and improves response time.

In 2019, they proposed the new breed of algorithm known as JAG_TDB_CC concurrency control algorithm in [5] which is a unique blend of a timestamp and lock-based concurrency control mechanisms. It reduces the risk of locks, frequent blocks, and long waiting times between different user sessions.

the proposed algorithm in[6]Petri net model of the locking protocol 2PL for concurrent transaction database, and on this basis, gives a deadlock-proof scheduling method using this protocol. This method is especially suitable for deadlock prevention of distributed concurrent transaction databases.

3. Algorithms to Secure Concurrency Control

There are different methods to concurrency control.

- Distributed Two-Phase Locking Protocol
- Timestamp-Based Protocols
- Validation-Based Protocols

but we have concentrated in this paper on one mechanism which is 2PL.

3.1. Locking

There are many ways to lock data:

1. Shared lock: It is denoted by S, if transaction T has obtained a shared lock on data item Q, then T can read but can't write Q. It is also called Read-lock[7].

2. Exclusive lock: It is denoted by X, if a transaction T has mode lock on data item Q then T can both read and write Q. It is also called Write–lock[7]

A. Two-phase Locking Protocol (2PL)

Two-phase locking protocol utilizes locks that block other transactions from accessing the same data during a transaction's life. Two phase locking protocol ensures serializability this protocol required that each transaction issue lock and unlock requests[7].
Two phase locking protocol are as follows:

1. Growing phase: called as growing phase in which the transactions can acquire or upgrade the locks[8]

2. Shrinking phase: the transactions can release or degrade the locks only[8], but may not obtain any new locks. It is also called contracting phase.

B. Strict Two-phase Locking (S2PL)

Strict 2PL protocol is an important variation of 2PL protocol. strict 2PL requires that in addition to locking being two-phase, all exclusive-mode locks taken by a transaction must be held until that transaction commits[9].

C. Conservative Two-phase Locking (C2PL)

C2PL's transactions obtain all the locks they need before the transactions begins. This is to ensure that a transaction that already holds some locks will not block waiting for other locks. Conservative 2PL prevents deadlocks[7].

	Locking Protocol	
	Advantages	Disadvantages
Two-phase locking protocol	- guarantee transaction serializability. - achieves a high degree of locking overall large area of the database.	- Occurs deadlocks among transactions. - Provides many rollback transactions.
Strict Two-phase locking protocol	- Avoids cascading abort.	- transactions may end up in waiting - Does not guarantee freedom from deadlocks.
Conservative Two-phase locking protocol	- Avoids deadlocks.	- doesn't prevent Cascading rollbacks. - dirty reads

Figure 1: Comparison between the algorithms

Figure 2 shows an 2PL is an algorithm that ensures serializability but does not prevent blockages and aborted transactions.
So, to fix these problems, there are two variants that are invented of 2PL protocol. the first one is strict 2PL which prevents cascading aborts but does not avoid blockages.
the second is Conservator 2PL which avoids blockages but does not ensure a good level of committed transactions and dirty reads.

III. PROPOSED WORK

1. Objectif of Our Proposed Mechanism

we propose a pessimistic concurrency control mechanism to be applied on real-time distributed database systems.

2. CS-2PL Mechanism

The CS-2PL algorithm combines the approach of Conservative two-phase locking and Strict Two-Phase Locking. The algorithm is applied by means of a trigger that allows a user session of a distributed transaction to perform its operations or make one transaction wait until the other transaction has released according to the rules and conditions specified in the algorithm.

This protocol guarantees the serializability between transactions performing read and writes operations, protects other conflicting user sessions from entering a standby state and permits them to stay inactive until the lock is released and It ensures that the schedule generated would be Cascade-les and prevents dirty reads and write..

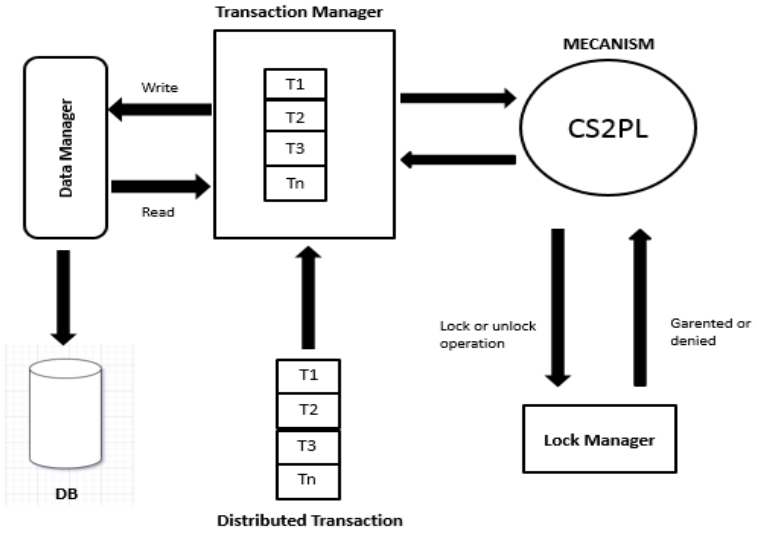

Figure 2: CS-2PL Mechanism

Figure 2 shows an architecture of SC-2PL which deals with serialization i.e. a technique that allows a transaction to be temporarily stopped while another transaction accesses the data, and shows how the mechanism works.

The Transaction Manager allows many user sessions of a distributed transaction to run simultaneously. And SC2PL is concurrency control mechanism solves conflicts between transactions in the event that multiple user transaction sessions attempt to access the same database resource at the same time. The lock manager is in charge of locking and unlocking different user transactions. Data manager takes care of the processing of the data in the database.

3. Concept of CS-2PL Mechanism

To explain the mechanism, we have given this example, which can be seen in figure 3 below.

the example like the following:

we consider as if we have a banking system and we want to carry out 4 transactions simultaneously on distributed databases and in the meantime, we want to ensure data integrity

T1: transfer from account A to account C $300

T2: transfer from account B to account A $200

T3: add $400 to account D

T4: Remove $100 from account C

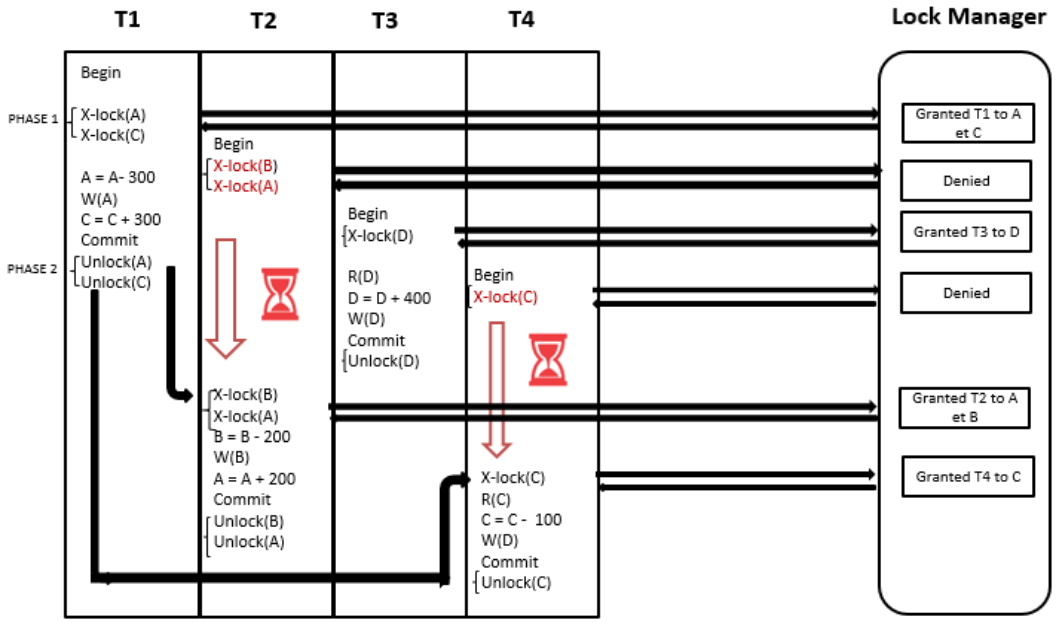

Figure 3: Concept of CS-2PL Mechanism

this mechanism consists of two phases:

PHASE 1:

This phase requires that the transaction locks all the items it needs from the Lock Manager before it starts executing the transaction by declaring it read and write beforehand. If any of the pre-declared required items cannot be locked, the transaction does not lock any of the items, but waits until all items are available to be locked. This is what make it deadlock-free as shown in Figure 3 below.

If a transaction asks Lock Manager to lock an entity, and the lock has been given to another transaction, the requesting transaction must wait.

PHASE 2:

All exclusive locks(X) or shared-locks held by the transaction will be unlocked until the transaction is committed.

once the transaction has been committed Lock Manager automatically takes care of unlocking the items that have been locked by this transaction. afterwards, the transaction waiting for these elements to be unlocked will automatically take over the locking of the items that are already pre-declared.

3.1. *Role of each phase*

PHASE 1: avoids deadlocks

A deadlock is a state where some processes request for some resources but those resources are held by some other processes[10], this situation occurs in concurrent programming when a set of transactions enter into infinite waiting situations.

but since the transaction has had all the locks when it needs it before its execution, therefore it will not have locks but it will have only a few waits of some transactions.

PHASE 2: avoids cascading abort and dirty reading and writing

dirty reads are a problem that occurs when a transaction reads data written by an uncommitted concurrent transaction and the same thing for dirty write.

Cascading aborts and dirty readings or writings are problems that occur when executing transactions simultaneously, but Phase 2 of our mechanism is designed to solve these problems. In this phase CS-2PL requires that each transaction must first be committed before releasing their locks.

IV. CONCLUSION

An important objective of a distributed database system is to ensure data consistency and integrity, and Concurrency control is a very major problem in the design of distributed database systems.

This paper presented an algorithm CS-2PL based on C2PL and S2PL which applies in situations with a high degree of simultaneity to attain non-blocking

locking, which improves the processing of simultaneous transactions. It removes the blocking problem of locking algorithms such as S2PL, and prevents cascading rollbacks that are found in C2PL. and it eliminates dirty reading and writing.

V. REFERENCES

[1] S. Vasileva and A. Milev, "Simulation Studies of Distributed Two-phase Locking in Distributed Database Management Systems," *Information Technologies and Control*, vol. 13, no. 1–2, pp. 46–55, Jun. 2015, doi: 10.1515/itc-2016-0010.

[2] M. Haroon, "Challenges of Concurrency Control in Object Oriented Distributed Database Systems," p. 6.

[3] Q. Abbas, H. Shafiq, I. Ahmad, and S. Tharanidharan, "Concurrency control in distributed database system," in *2016 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, Jan. 2016, pp. 1–4, doi: 10.1109/ICCCI.2016.7479987.

[4] M. Mohamed, M. Badawy, and A. EL-Sayed, "An improved algorithm for database concurrency control," *Int. j. inf. tecnol.*, vol. 11, no. 1, pp. 21–30, Mar. 2019, doi: 10.1007/s41870-018-0240-y.

[5] J. A. Gohil, K. A. Popat, and P. M. Dolia, "Comparative Study and Performance Evaluation of JAG_TDB_CC Concurrency Control Algorithm for Temporal Database," p. 6.

[6] Y. XiaoLing, "A Deadlock Prevention Algorithm for The Two-Phase Locking Protocol Based on Petri Net," in *2019 6th International Conference on Systems and Informatics (ICSAI)*, Shanghai, China, Nov. 2019, pp. 889–892, doi: 10.1109/ICSAI48974.2019.9010538.

[7] U.-M. A. K. Srivastava, B. Singh, A. Singh, N. Singh, and P. Singh, "Concurrency Control in Distributed database management systems: Dealing Unfair Transactions at Higher Access Classes," vol. 07, no. 03, p. 8, 2019.

[8] . M. K. G., . R. K. A., and . B. S. B., "Study of Concurrency Control Techniques in Distributed DBMS," *IJMLNCE*, vol. 2, no. 4, Dec. 2018, doi: 10.30991/IJMLNCE.2018v02i04.005.

[9] V. Verma, G. Verma, D. Sisodia, and P. Vashist, "Strict 2 Phase Locking in Organizational Data Protection," vol. 1, no. 2, p. 4.

[10] P. Tomar and M. Bhardwaj, "A Review on Deadlock Detection in Distributed Database," *Advances in Computer Science and Information Technology*, vol. 2, no. 8, p. 3, 2015.