Scientific Journal of Faculty of Education, Misurata University-Libya, Vol. 1, No. 10, Mar. 2018

Published on Web 01/3/2018

المجلة العلمية لكلية التربية، جامعة مصراتة، ليبيا، المجلد الأول ـ العدد العاشر، مارس 2018م

Transaction Management and Concurrency Control

Shaimaa Falah Hamied Amal abdulsalam ARhoma
Nadia abdulsalam Alhaweiaj

Abstract:

Evolution in distributed systems applications is increasing day after day, accompanied by the emergence of many of the problems that are looking for a solution, in addition to the need of technologies which controls the distributed environment. This paper addressed concepts of transactions and focus on deadlock which is one of the common problems that caused transaction failures . Deadlock prevention is one of the deadlock types that will be discussed then followed by presenting the Banker's algorithm provided with an example as a solution for this type .

1. Introduction

In past computer systems were a single and performing services independent of other computers , so the person who is the user 's computer he is only the one that can utilizing and controlling the information in this computer and cannot someone else in another place to enter information in this device. Nowadays many systems become distributed and users can be allowed access to a shared database. The increase in the number of operations on a shared database caused many problems that need to find techniques to solve, such as operation conflicting and deadlocks. So we need a control and management tools to ensure a success and security environment for these operations and shared data . When a sequence of operations are run on database and commit or abort as a single

atomic unit this named transaction , which is contain one or more of the

Process that are working together to either completed transaction to succeed or abort to cause the fail transaction.

Concurrency control is One of the techniques to be addressed by this paper, it is responsible for concurrent transaction, and permits users to access database in a safe state. This paper will discuss some of technique in distributed system and it issues such as deadlocks

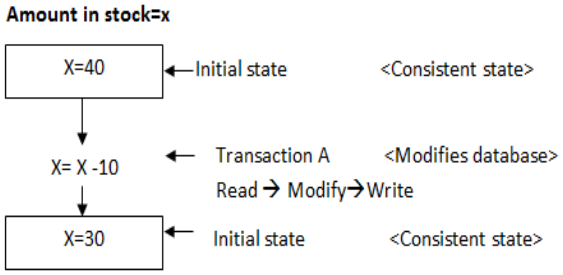
Problems and suggested solutions by using algorithm to solve this problem.

2. Transaction

Transaction can be define as a Sequential executions of operations which occur within computer applications and is directed by the application user who starts the transaction and terminates it .

As an example of transaction is the use of ATM application for money withdraw, where the user request for a certain amount of money and then withdraw. Figure (1) shows an example of transaction, in the beginning the initial state is at x=40, and after sequence of modifications on x, resulting final state at x=30. Transaction works as an atomic

unit, it is either be successfully completed if the process have been completed or it is fail due to abort processes which are abort for one reason or another . This lead some sources to define Transaction: "an atomic



Unit of consistency and recovery" [1]. Database transactions have common properties denoted by *ACID*, which are the acronym of the following properties:

Atomicity: it is mean that all parts of database transaction must be completed or the transaction is aborted if incomplete due to any possible reasons.

Consistency: relates with database consistent. It means the database must be in a consistent state before and after the transaction, and database transaction must not break the database integrity constraints.

Isolation: means ensuring that data of current transaction database is not be accessible or usable by another transaction until the finishing of the transaction currently executed.

Durability: means save all the changes of data or data updates which resulting from execution of a transaction and permanently so that to ensure there is no data lose even in case of a system failure after the transaction has been completed. Serializability is also an important property which means if transaction executed one after one, we will get similar results whether if the transaction executed concurrently.

3. Concurrency control

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" [2].

Concurrent transactions may be conflict, and Simultaneous executions of transactions over multi user database environment may effect on data integrity and consistency, and causes several problems such as lost updates, uncommitted data and inconsistent retrievals. By applying concurrency control techniques that provide a serialization order among conflicting transactions, we can prevent this conflict which causes an incorrect overall result, even if each transaction is correct when executed in isolation. "Serializability is a widely accepted correctness criterion for controlling concurrent execution of transactions in database systems Serializable schedules provide correct results and leave the database consistent" [3]. Many methods were

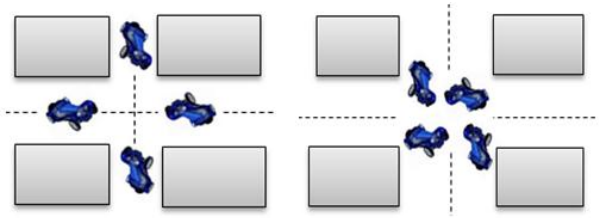
found for concurrency control mechanism such as Locking, Timestamps and Optimistic Concurrency Control. A *lock* is a mechanism used to preventing conflicting operations in case of two or more users are available and they are using the same data Timestamps, all lock of information is managed by a lock manager. *Timestamping* means assign a timestamp to each transaction, and Record the timestamps of transactions that last read and write each database element. In case of conflicting transaction, instead of using locking and timestamping, Before committing each transaction examine their data and verify whether other transaction have modified its data ,if the check reveals conflicting modifications, the committing transaction rolls back. And This is named *optimistic concurrency control*.

Let us take an example of concurrency control, suppose that there are two users, every one of them is trying to update the same data at the same time. Concurrency control responsibility here is to prevent one user from seeing out-of-date data while another user is updating the same data.

4. Deadlock

Generally, deadlock exist where there are many processes competing on some resources and each one is trying to hold these resource and waiting for the other to release the resource ,but no one hold it.

In an operating system , deadlock occurs, when a process or thread in the state of waiting for the source requested , which is also held by another waiting process trying to get it but is waiting for another resource . If a process is unable to change its state and cannot get to the source, who requested by another process, then the system is said to be in a deadlock .



(a) *Deadlock possible*
(Before deadlock occurrence)

(b) *Deadlock*
(Deadlock state)

Figure (2) Example for the deadlock idea
Four processes compete for one resource

Figure 2(a) and (b) illustrates an example for the deadlock idea. There are many conditions that must be available to cause a deadlock occurrence:

- ***Mutual Exclusion***: At least one unsharable resource - processes claim exclusive control of resources they need.
- ***Hold and Wait***: Process holds one resource while waiting for another.
- ***No Preemption***: Resources only released voluntarily - no interruption possible, it is mean forcefully withdrawn by another process.
- ***Circular Wait***: Circular chain of processes - each waiting for a resource held by another.

4.1. Deadlock detection

This type is based on the idea of allowing deadlock to occur, and then the system can detect the presence of the deadlock in the transaction and then correct it by using one of the methods such as Process termination and resource preemption. After detect deadlock, the transaction is aborted and all the changes made by this transaction are rolled back and all locks are released [7].

4.2. Deadlock prevention

If the system know about the possibility of deadlock in one of transactions, it can abort any transaction that requesting a new lock, the changes made by this transaction are rolled back and rescheduled, all locks are released and the other transactions are continue. That is mean we avoid the conditions that lead to deadlocking .Deadlock prevention can be achieved by using non-blocking synchronization algorithm which is removing the mutual exclusion condition, means that no process will have exclusive access to a resource. Also such serializing tokens algorithm is used to removing the hold and wait or resource holding [8]. Dijkstra's algorithm can be used to avoid circular

waits include disabling interrupts during critical sections and using a hierarchy to determine a partial ordering of resources [6].

4.3. Deadlock Avoidance

If we can detect deadlock transaction, we can avoid it. In this type ,the system checks whether the next state will be safe or unsafe , it can do this by knowing in advance at any time many thinks : resources currently available, resources currently allocated to each process and resources that will be required and released by these processes in the future . The system then only grants requests that will lead to safe states [4]. The transaction must obtain all the locks it needs before it can be executed, this technique avoids rollback of conflicting transactions by requiring that locks be obtained in succession. The Banker's algorithm is one of the good solution that used for deadlock avoidance, it requires resource usage limit to be known in advance [4].

4.3.1. Banker’s Algorithm

The Banker's algorithm is used to prevent deadlock by denying or postponing the request of process. It is called banker because its idea is similar to the idea of bank, where the customer who is the account holder , he can not withdraw the a mount of money that exceed the current account value . This algorithm is developed by Edsger Dijkstra , it was developed in the design process for the operating system and originally described (in Dutch) in EWD108. The name is by analogy with the way that bankers account for liquidity constraints [5].

Whenever a process requests resources, the Banker's algorithm is run by the operating system. The idea of this algorithm is depend on check the possibility of deadlock occurrence, if the algorithm determine s that the system is in unsafe state; this means there is a risk of deadlock. To avoid this deadlock, the algorithm is denying or postponing the request. In case of the determinations were checked that system is in safe state, the algorithm allows request because they do not lead to deadlock. Thus we can ensure that the transactions are safe from deadlocks. The process that gets all its requested resources it must return them in a finite amount of time.

The Banker's algorithm needs many thinks to know ,these are : the number of each resource that each process could possibly request, the number of each resource that each process is currently holding and the number of each resource that each process that the system currently has available .

The Banker's algorithm checks the request and comparing it with the both of max and available, as following:

1. Request ≤ max, else set error condition as process has crossed maximum claim made by it.
2. Request ≤ available, else process waits until resources are available.

4.3.2. Banker's Algorithm structures

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

- Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type Rj available.
- Max: An n×m matrix defines the maximum demand of each process. If Max [i,j] = k, then Pi may request at most k instances of resource type Rj.
- Allocation: An n×m matrix defines the number of resources of each type currently allocated to each process. If Allocation [i,j] = k, then process Pi is currently allocated k instance of resource type Rj.
- Need: An n×m matrix indicates the remaining resource need of each process. If Need [i,j] = k, then Pi may need k more instances of resource type Rj to complete task.
  Note: Need = Max – Allocation

4.3.3. Banker's Algorithm Example

In this example we have 5 processes Po through P4 and 3 resource types A (10 units), B (5 units), and C (7 units), as shown in in table (1). Suppose Snapshot at time To, each process's state is as shown in the figure 4 (a). By subtracting the allocation values from max values , we

obtain the need values as shown in table(2) , then rearrange the table as in table(3). The system is in a safe state since the sequence:
< P1, P3, P4, P2, Po> satisfies safety criteria.
After that we check the system state whether it is safe or unsafe depending on the coming requests. If p1 request is (1,0,2) **,** compare it with Available as follow :

Check that Request ≤ Available , that is (1,0,2) ≤ (3,3,2) ⇒ true. And the system is in safe state, so p1 hold the resource. Now re calculate the new Available as follow:

Check that Request ≤ Available , that is (1,0,2) ≤ (3,3,2) ⇒ true, and the system is in safe state , so p1 hold the resource . Now re calculate the new Available as follow:

Available = (3,3,2) - (1,0,2) ⇒ (2,3,0) **.**

In case of the request is (3,3,0) by p4 :

Request (3,3,0) ≤ Available (2,3,0) ⇒ false, here the system is in unsafe state, so the request is impossible and must either be denied or put on a waiting list .

Table (1) Existing processes and resources

Process	**Allocation** **A B C**	**Max** **A B C**	**Available** **A B C**
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Table (2) Calculate the need values

Process	Need
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Table (3) Calculate new value of available

Process	Allocation A B C	Max A B C	Need A B C	Available A B C
P_0	0 1 0	7 5 3	7 4 3	2 3 2
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

5. Conclusion

Most of distributed computer systems carried out one or many transactions, which are either interdependent or concurrent. The reliable transaction systems have a common property under the acronym ACID: atomicity, consistency, isolation, and durability. Concurrent transactions can be manage by Concurrency control techniques such as Locking, Timestamps and Optimistic Concurrency Control, which are providing a serialization among conflicting transactions and preventing this conflict.

One of the common problem that happened in transaction is the deadlock, which occurs when four conditions are met: a mutual Exclusion, hold and wait, no preemption and circular wait. Many

techniques were been studies to control deadlocks such as deadlock detection, deadlock prevention and deadlock avoidance.

We addressed deadlock avoidance and discussed The Banker's algorithm to solve deadlock problem. As long as the Banker's algorithm run in the operating system, we can prevent deadlock from occurrence. The algorithm is checking the requests of processes, determines whether the system is in safe or unsafe state, and avoids deadlock by denying or postponing the requests.

6. References

[1] Machigar Ongtang1, Ali R. Hurson1, Yu Jiao2, and Thomas E. Potok." Agent-based Transaction Management for Mobile Multidatabase", Dept. of Computer Science and Engineering, Penn State University Computational Sciences and Engineering Division, Oak Ridge Natl. Laboratory, page: 1. November 2007.

[2] philip a. bernstein and nathan goodman," Concurrency Control in Distributed Database Systems", Computer Corporation of America, Cambridge, Massachusetts 02139. Computing Surveys, Vol. 13, No. 2, June 1981.

[3] Ersan Kayan, " Real-Time Transaction Management in Mobile Computing Systems", Department of Computer Engineering and Information Science, Bilkent University Bilkent, Ankara 06533, Turkey, page: 2.

[4] Silberschatz, Abraham (2006). Operating System Principles (7 ed.). Wiley-India. P. 237. Retrieved 29 January 2012.

[5] E. W. Dijkstra. "Selected Writings on Computing: A personal Perspective", Springer-Verlag, New York Inc. 1982.

[6] Manisha Mohanty, Prerna Kumara. "Deadlock Prevention in Process Control Computer System". International Journal of Computer Applications (IJCA) (0975 – 8887), 2013.

[7] Jerzy Brzezinski, Jean-Michel Helary, Mukesh Singhal." Deadlock Models and A General Algorithm for Distributed Deadlock

Detection".Journal of Parallel and Distributed Computing,Volume 31, Issue 2, December 1995, Pages 112-125.

[8] Mahboobeh Abdoos," Improved Deadlock Prevention Algorithms in Distributed Systems",International Journal of Engineering and Applied Computer Science (IJEACS) Volume: 02, Issue: 02, February 2017.