Illustration of Timestamp Ordering in Controlling Concurrency in Distributed Database

Rinki Chauhan,
Department of CSE
MRIU, Faridabad
iknirs@gmail.com,

Preeti Bhati
Department of CSE
MRIU, Faridabad
versatilepreeti1986@yahoo.com

***Abstract*—A distributed database consists of different number of sites which are interconnected by a communication network. In this environment in absence of proper synchronization among different transaction may lead to inconsistency of databases. In this paper we are going to discuss various timestamp ordering algorithms for concurrency control. These algorithms are also illustrated with an example.**

Keywords—transaction, concurrency control, timestamp, distributed database

I. INTRODUCTION

Distributed database systems (DDBS) are systems that have their data distributed and replicated over several locations or sites unlike centralized databases where one copy of the data is stored. Both types of databases have same problem of access control, such as concurrent user access.

Concurrency control is a method of managing concurrent access of transactions on a particular data item such that the consistency of the database is maintained. Consistency means when any transaction comes for execution the database is in consistent state and when it leaves the system the database should be in consistent state. Also, the result produced by the transaction should be correct. This problem becomes complex in distributed databases since the data is not stored at one place. The user can access the data from any site and the controlling mechanism at other site may not recognize it instantly. Various algorithms have been designed for controlling concurrency of the database.

The notion of a transaction is of fundamental importance to concurrency control. In a distributed database environment, a transaction may access data stored at more than one site. Each transaction is divided into a number of sub-transactions, one for each site at which data accessed by the transaction is stored.

II. DISTRIBUTED TRANSACTION-PROCESSING MODEL

For understanding how a concurrency control algorithm operates we present a simple model of Distributed Database Management System in this section in Fig. 1. A distributed database management system(DDBMS) is a collection of sites interconnected by a network.

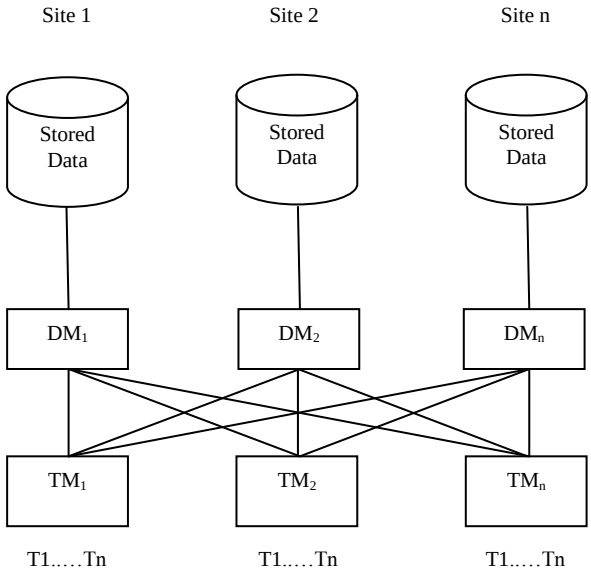

Figure. 1. DDBMS Transaction Processing Model

Each site running on one or both of the following software modules: a transaction manager (TM) and data managers (DM). TM supervises the interactions between users and DDBMS while DMs manage the actual database. Here, the communication network is considered to be reliable. This means that if site A sends a message to site B that message reaches to site B without any error or vice-versa.

III. TRANSACTION SYNCHRONIZATION BASED ON TIMESTAMP ORDERING

Locking algorithms are considered best for concurrency control in a database. But locking algorithms may lead to deadlock situations in some cases which will require deadlock detection, prevention and avoidance mechanisms. Such complications are eliminated in timestamp ordering algorithms. In timestamp ordering [2], we use a unique identifier called timestamp assigned by the Transaction Manager to each transaction for execution. The TM attaches the timestamp to all dm-reads and dm-writes issued on behalf of the transaction. DMs process conflicting operations in the timestamp order.

Operations are conflicting if they are accessing the same data item and one of them is a write operation. Thus there are two kinds of synchronizations needed: read-write (rw) synchronization and write-write (ww) synchronization.

In rw synchronization, both the operations are trying to access the same data item and one of them is write operation and the other one is a read operation.

In ww synchronization, both the operations are trying to access the same data item and both of them are write operation.

A. Implementation of Basic Timestamp Ordering

Basic T/O[4,5] is quite simple. For every data item at every DM there is a timestamp maintained which is the largest timestamp of the transaction (dm read or dm write) which has last used the data item. They are called as the read and write timestamps if the transactions are dm read and dm write respectively.

In rw synchronization, if a dm read comes with read operation on data item x denoted as r(x) with a timestamp TS then check if TS is greater than the write timestamp of x or not, i.e. TS>Wts(x). If TS is smaller, then abort the issuing transaction otherwise accept and execute r(x) and set the read timestamp of x as timestamp greater than between TS and Rts(x)(i.e. read timestamp of x). Similarly, if a dm write comes with a write operation on a data item x denoted as w(x) with a timestamp TS then check TS>Rts(x). If TS is smaller, then abort the issuing transaction else accept and execute w(x) and set the write timestamp of x as greater between TS and Wts(x).

In ww synchronization if a dm write arrives for a data item x with a timestamp TS where TS<Wts(x) then the write is ignored else the dm write is accepted and the timestamp of x is set as TS.

B. Multiversion T/O

In Multiversion Timestamp ordering[2,5], instead of maintain a largest timestamp for every data item we maintain a set of read timestamps and a set of write timestamps called as versions.

In rw synchronization, if a dm read arrives on a data item x with a timestamp TS then the largest write timestamp or version of x is searched which is less than the TS. This version is read by that dm read and a new read timestamp, TS is added to the set of read timestamps of x. Similarly, if a dm write comes on a data item x with a timestamp TS then interval from TS to the smallest version of x greater than TS is searched, if in that interval any read timestamp of x lies then the dm write is rejected else it is executed and a new version of x is created as TS.

In ww synchronization, a dm write creates a new version of timestamp and is never rejected.

C. Conservative T/O

This algorithm[5,7] is used to minimize the number of restarts. If any operation arrives which can cause a restart in future is delayed until no restarts are possible in future.

Each site maintains several pairs of queues, one read queue and one write queue in timestamp order for every site in the network. Each read queue contains requests for read operations on local data from transaction originating at a remote site, while each write queue maintains the same information for update requests. Individual read and update requests are labeled with the timestamp of the transactions which issued them.

If a site B issues a dm read request to the data manager at site A. DM at site A will insert the request in its read queues at proper place according to the timestamp. The read operation by the DM at A after checking and ensuring that there is no operation in the queue having a younger timestamp. If any such operation is present then the read request will have to wait until its timestamp becomes the youngest one.

Same goes for the write operation. Any dm write coming from site B at site A will be placed in the write queue of site A as per the timestamp of the operation. After checking that the write queue at the data manager is not empty and the timestamp of the write request is the youngest one, the request is executed else the request has to wait until it becomes the youngest one. If the queue is empty then a null request can be sent from the site.

The drawback with this algorithm is that that a site blocked due to an empty queue could issue a request having a smaller timestamp later.

IV. EXAMPLE

In our example we have taken a scenario in which there are three sites on which three data items(a, b and c) are stored a at site1, b at site2 and c at site3 respectively.

Three transactions are issued as mentioned below:

T1 : r1(a)r1(b)w(b) at site 1 having timestamp 1

T2 : r2(b)w2(b) at site 2 having timestamp 2

T3 : r3(c)w3(c)r3(a)w3(a) at site 3 having timestamp 3

Now, for executing T1 it has to be divided into two subtransactions by transaction manager namely :-

T11 : r1(a) to be executed by the DM at site 1 and

T12 : r1(b)w(b) to be executed by the DM at site 2

T2 will get executed at site 2 and T3 will again be divided into two subtransactions namely :-

T31 : r3(c)w3(c) to be executed by the DM at at site 3 and

T32 : r3(a)w3(a) to be executed by the DM at at site 1

Now as per the timestamps issued, the transactions to be executed at site 1, site 2 and site 3 are :-

At site 1 : r1(a)r3(a)w3(a),

At site 2 : r2(b)w2(b)r1(b)w1(b),

At site 3 : r3(c)w3(c) respectively.

In basic timestamping algorithm, if the data item a is having its read timestamp as 1 and write timestamp as 1, data item b is having its read timestamp as 2 and write timestamp as 1 and the data item c is having its read timestamp as 3 and write timestamp as 2.

At site 1 write timestamp of a is not greater than the timestamp of the dm-read of site1 so it will get executed but since the timestamp of the dm read same as the read timestamp of the data item a so the read timestamp of a will remain 1 only. Now, for r3(a) the timestamp of the dm-read is 3 and the write timestamp of a is 1 so it will also get executed and since the read timestamp of a is less than timestamp of dm-read so the read timestamp of a will now become 3. Now, for w3(a) the timestamp of dm-write is not less than the read timestamp of a i.e. 3. Also, the write timestamp of a is also not greater than the timestamp of dm-write so it will get executed and the write timestamp of a will become 3.

At site2 timestamp of dm-read is not less than the write timestamp of the data item b so it will get executed and the read timestamp of the data item will remain 2. For w2(b) the timestamp of dm-read is not less than the read timestamp of the data item b and also, the write timestamp of the data item is less than the timestamp of the dm-write so it will get executed and the write timestamp of the data item will become 2. r1(b) will get aborted since its timestamp is less than the write timestamp of the data item b. similarly, w1(b) will also get rejected since its timestamp is less than the read timestamp of b.

At site 3 r3(a) will get executed since its timestamp is not less than the write timestamp of data item c and the read timestamp of c will remain 3. Also, w3(c) will be executed since the timestamp of the dm write is not less than the read timestamp of the data item. Also, the write timestamp of c is less than the timestamp of the dm-write. So, write timestamp of c will now become 3.

In multiversion timestamping algorithm, if the set of read timestamps of data item a are 3,5 and 6 and different versions of a at site 1 are 0, 1, 2, 4and 6. At site 2 the set of read timestamps of data item b are 0 and 6 and different versions of b are 0, 1, 3, 4and 5. At site 3 the set of read timestamps of data item c are 1, 3,5 and 6 and different versions of c are 1, 2, 4, 5and 6.

At site 1 r1(a) will get executed and will get the value of a at version zero. Also, a new read timestamp will be created. Now r3(a) will also get executed and will read the value of a at version 2 thereby creating a new read timestamp 3. w3(a) will not be executed since there lies read timestamps of a between the interval of 3-6.

At site 2 r2(b) gets executed and will get to read the value at version 1 and will be creating a timestamp 2. w2(b) will be executed since there is no read timestamp between the interval 2-5. r1(b) will get executed thereby creating a new read timestamp 1 and will get to read the value at version 0. w1(b) will be executed since there is no read timestamps between interval 1-5

At site 3 r3(c) gets executed and will read the value at version 2 and will create a new read timestamp 2. w3(c) will not be executed since there are many read timestamps between interval 3-6.

In conservative timestamping, every transaction manager will maintain a queue for every site and execute them in increasing order of the timestamps. So, transaction manager at site 1 will have r1(a) in queue Q11 r3(a)w3(a) in queue Q13. TM at site 2 will have r1(b)w1(b) in Q21 and r2(b)w2(b) in Q22 and at site 3 TM will have r3(c)w3(c) in Q33.

At site 1 since r1(a) is having smallest timestamp so it will be executed first then r3(a)and w3(a) will be executed.

At site 2 r1(b) is having the youngest timestamp so it will be executed first followed by w1(a) then r2(b) and then w2(b) will get executed.

At site 3 first of all r3(c) will execute and then w3(c) will be executed.

V. CONCLUSION

In this paper, we have presented a distributed transaction processing model. We have discussed almost all types of timestamp ordering algorithms for concurrency control in distributed databases. We have considered the problem of deadlocks faced by the locking algorithms which is eliminated in timestamp ordering algorithms.

We have left one important issue as performance. The performance factor of concurrency control algorithms depends on system throughput and transaction response time

The performance analysis of algorithm still remains to be discussed. We hope, and really recommend, that our future work on distributed concurrency control will concentrate on the performance of algorithms.

REFERENCES

[1] C. J. Bouras P. G. Spirakis "Performance Models for Perfect and Imperfect Clocks on Timestamp Ordering in Distributed Databases" MASCOTS '93 Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems

[2] F. Bukhari and Sylvia L. Osborn "Two Fully Distributed Concurrency Control Algorithms" IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 5, NO. 5, OCTOBER 1993

[3] Li Victor O. K., "Performance Models of Timestamp Ordering Concurrency Control Algorithms in Distributed Databases", IEEE Transactions on Computers, Vol. C–36, No. 9, September 1987, pp. 1041– 1051

[4] Singal M. , "Performance analysis of the basic timestamp ordering algorithm via Markov modelling", Performance Evaluation 12(1991), pp 17–41

[5] Tamer Ozsu "Principles of Distributed Database Systems" Prentice Hall 1999.

[6] UGUR HALICI and ASUMAN DOGAC "Concurrency Control in Distributed Databases Through Time Intervals and Short-Term Locks" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. VOL. 15. NO. 8 AUGUST 1989.

[7] Wang C., Li V., "Queueing analysis of the conservative timestamp ordering concurrency control algorithm", Proceeding IEEE International Computing Symposium 1986, pp. 1450–1455