Timestamp Based Optimistic Concurrency Control

Conference Paper · December 2005

DOI: 10.1109/TENCON.2005.300909 · Source: IEEE Xplore

2 authors:

Quazi Mamun
Charles Sturt University
57 PUBLICATIONS **107** CITATIONS
SEE PROFILE

Hidenori Nakazato
Waseda University
56 PUBLICATIONS **104** CITATIONS
SEE PROFILE

Timestamp Based Optimistic Concurrency Control

Quazi Ehsanul Kabir Mamun
Global Information and Telecommunication Studies
Waseda University
Tokyo, Japan
mamun@fuji.waseda.jp

Hidenori Nakazato
Global Information and Telecommunication Studies
Waseda University
Tokyo, Japan
nakazato@waseda.jp

***Abstract*—Optimistic Concurrency control demonstrates a few improvements over pessimistic concurrency controls like two-phase locking protocol or time-stamp based protocol. But the price of coarse detection of conflicts may sometimes be high and consequently discounts the advantage of optimistic concurrency control protocol. In this paper we have reduced the space of coarse detection of conflicts of optimistic concurrency control protocol.**

I. INTRODUCTION

Transaction processing, concurrency control and recovery issues have played a major role in conventional databases [1], and hence have been an important area of research for many decades. For the concurrency control algorithm, there are two possible views: the pessimistic view that many transactions will conflict with each other, or the optimistic view that not too many transactions will conflict with each other. Pessimistic methods synchronize the concurrent execution of transactions early in their execution and optimistic methods delay the synchronization of transactions until their terminations.

Under lock-based pessimistic concurrency control a transaction can obtain a lock only if another transaction does not hold a conflicting lock on the same data item. When a lock is set, other transactions that need to set a conflicting lock are blocked until the lock is released, usually when the transaction is completed. The more transactions that are running concurrently, the greater the probability that transactions will be blocked, leading to reduced throughput and increased response times. The resource costs involved in lock maintenance are considerable and these costs dramatically escalate as the number and complexity of concurrent transactions increases. However, there are always a large number of incidences where this overhead is totally wasted because the transactions do not actually conflict with one another. The effects of locking strategies are difficult to predict and the consequences are often only apparent when an application is put into production. Vendors have to provide a vast number of tuning mechanisms to optimize performance which increases the complexity and cost of the development and maintenance of the database system.

On the other hand, optimistic concurrency control is based on the premise that it is sometimes easier to apologize than to ask permission [2]. This gives a great advantage since in practice, conflicts are relatively rare [3]. The big advantages of optimistic concurrency control are that it is deadlock free and allows maximum parallelism because no process ever has to wait for a lock.

In this paper we have presented a variant of optimistic concurrency control protocol that shows better performance than original optimistic concurrency control protocol (OCCP). This paper has been organized as follows – in section 2 we have presented the original optimistic concurrency control. The following section shows some problems along with original OCC. We have presented our timestamp based OCC in section 4 and the proof of correctness is given in section 5. Comparison of our algorithm with OCC and timestamp based concurrency control is discussed in section 6. Finally we have drawn conclusion in section 7.

II. ORIGINAL PROTOCOL

In optimistic concurrency control protocol (OCCP), the basic idea is to be as free as possible to let the transactions execute since the presumption is that most transactions do not conflict with each other. If there are fewer conflicts then OCCP would work more efficiently than locking based protocols. Optimistic algorithms view a transaction consisting of read, validate and write phases. In OCCP, the transactions are validated just before the write phase when the modifications are to be written to the database. Thus the transactions are always allowed to initiate without any delay once submitted to the optimistic scheduler i.e. they are allowed to carry out their read phases. Each transaction reads the data, does the computations and makes local copies of its modifications in its private workspace area. After that the transaction need to be validated so as to check the compatibility of its modifications with the database. If they are compatible then changes are written otherwise the transaction is restarted.

The transaction proceeds in three phases namely Read, Validate and Write phases.

Read: The transaction reads values from the database and writes them into a private workspace.

Validation: When the transaction is about to commit the OCCP checks whether the committing does not conflict with other concurrent transactions. If a conflict exists then the

transaction is aborted and restarted. Its work space is also cleared.

Write: When the validation phase determines that the transaction does not conflict with other transactions then its private work space is copied into the database.

In OCCP only the transactions are given timestamps (TS) at the beginning of the validation phase. The validation criteria are used to determine the ordering of the transactions. For every pair of transactions Ti and Tj such that TS(Ti) < TS(Tj), one of the following validation criteria must satisfy [6]. Here $TS(T_x)$ is the TS of transaction T_x.

1. All the phases of T_i complete before T_j. Thus T_i and T_j are executed in serial order.

2. T_i ends before write phase of T_j, and T_i does not modify any database items read by T_j. Here even though T_j reads data when Ti is modifying they can be executed together since T_j doesn't read any data modified by T_i. T_j can modify objects already modified by Ti but it doesn't matter since Ti has already committed.

3. T_i finishes its read phase before T_j finishes its read phase and they both do not read/write any common objects. Here T_i and T_j can write objects simultaneously but since the data read/written is not common, the operation can be done simultaneously.

In order to validate T_j the above validation criteria must hold with respect to a transaction T_i where $TS(T_i) < TS(T_j)$. By this validation we just check that both the transactions do not conflict with each other and that they could be allowed to execute concurrently. The validation involves maintaining a list of objects read and written by each transaction. During the validation process other transactions can't commit since we still have to check the conflicts with recently committed transaction. When a transaction is validated then its write phase should be allowed to complete so that its modification would be viewed to the next transaction picked for validation. Thus it's obvious that only one transaction should be allowed to enter validation and write phase and the other transactions if any would have to be put on hold. This could be a bottleneck situation and it should be kept as small as possible to increase concurrency. One of the ways to do this could be to use indirection (switching pointers to the database items) while modifying the database in the write phase.

III. PROBLEMS OF OCCP

There are some simulation studies i.e., [10] – [23] which are quite positive for optimistic methods. Greater acceptance can only be achieved if some serious drawbacks of the original approach of Kung and Robinson [6] can be overcome. The following issues are essential:

1. The degree of potential parallelism of optimistic methods is high. However, the same is true for the risk of being restarted unnecessarily. It would be desirable to avoid restart in all cases where detected conflict actually does not endanger serializability.

2. Since long transaction run for a longer time than transactions of average size they have an increased risk of being subject to restart. However, long transactions should have similar chances of committing to short ones.

3. Since optimistic concurrency control relies on rollback as the means of synchronization transactions might be restarted repeatedly. This phenomenon is known as the starvation problem and, of course, must be prevented.

Transactions may be restarted unnecessarily, as figure 1 shows:

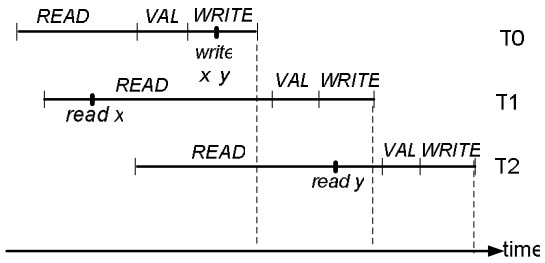

Fig. 1 Unnecessary restart by OCC

Transaction T0 writes during the read phases of T1 and T2 and thus has to be considered in their validation phases. Using the validation scheme of [6], both transactions have to be restarted. If we take a closer look at the above scenario the following becomes clear:

- T1 reads an object that will later be written by T0. Serialization in commit order is not possible. Hence this conflict is "serious": It signals a non-serializable situation.
- However, the conflict between T2 and T0 is not "serious", since the conflict due to *y* simply expresses the serialization constraint T0 →T2. Restart of T2 is unnecessary unless there is another object which T0 writes after T2 reads it.

IV. TIMESTAMP BASED OCC PROTOCOL

Like traditional Optimistic concurrency control protocol every transaction will go under three phases Read, Validation and Write phases. In the read phase transactions read data items and updates (pre-write) data items only in a private work space. If validation is successful, actual updates take place in the database. We assign some symbols to each transaction at various stages of the transaction shown in the following figure

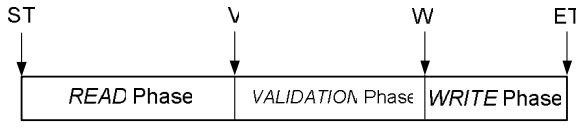

Fig. 2 Symbols assigned within a transaction

During the execution each transaction will read and/or write on some data items. These reads and writes are enlisted

in two sets – ReadSet and WriteSet. The members of ReadSet and WriteSet will be tuples of data items conjugated with the time instance the data item was accessed. That is ReadSet / WriteSet will be of the form

$\{(\alpha_m, t_n), (\alpha_p, t_q), \ldots \mid$ data item α_m was accessed at time t_n and so on $\}$.

Validating transaction will go through the following layer of checking sequentially. If condition 1 is true, validating transaction is verified. Otherwise both condition 2 and 3 should be checked for validity.

1. Is the validating transaction in serial position with conflicting transaction?
2. Is there any write-write conflict from validating to conflicting transactions?
3. Is there any read-write conflict from validating to conflicting transactions?

We set our conflict checking precise such that validating transaction should not be aborted for non-serious conflicts.

Let T_i and T_j be the transactions. The algorithm for validation of transaction T_j is depicted in Fig. 3. The notations of the algorithm require some explanations. 'Valid' is a variable that stores Boolean value. If the value of this variable remains 'TRUE', transaction T_j will be verified otherwise T_j should be aborted. $T_x(Y)$ means the time of Y stage in transaction T_x. Here Y may refer ST, V, W or ET as shown in Fig. 2. For example, $T_i(ET)$ means the time instance transaction T_i commits. $\varepsilon(T_x)$ refers to the time instance recorded in the ReadSet / WriteSet of transaction T_x for data item ε. For validation, transaction T_j will consider all transactions, say T_i for which $T_i(V) < T_j(V)$.

```
1  if (Ti(ET) < Tj(ST))  Valid = TRUE
2  else{
3        Valid =TRUE
4        if ((WriteSet (Tj)∩WriteSet (Ti) ≠ Ø) and
5                                          (Ti(ET) > Tj(V)))
6              Valid = FALSE
7        else if (ReadSet(Tj)∩WriteSet (Ti) = β) {
8              for every element ε of β {
9                    if  (ε(Tj) <Ti(ET)) {
10                         Valid = FALSE
11                         Break
12                         }
13                   }
14     }
15 }
```

Fig. 3 Timestamp based OCCP

V. PROOF OF CORRECTNESS

To determine whether a schedule is serializable, we construct its serialization graph. This graph is constructed as follows: The transactions {T1, T2, ...,Tn} are nodes in the graph. There is a directed edge from Ti to Tj if and only if, for some x, one of the following holds:

- ri[x]<wj[x],
- wi[x]<rj[x], or
- wi[x]<wj[x].

Here ri[x] means the time of read operation on data item x by transaction Ti, wj[x] means the time of write operation on data item x by transaction Tj etc.

We assume two transactions T_i and T_j. We also assume that $T_i(V) < T_j(V)$ and T_j has just arrived at validation phase. If the condition at line 1 in Fig. 3 is met, we can say that transactions T_i and T_j are not concurrent because one transaction starts after committing of another transaction. So, always the schedule will be a serial one.

If T_i and T_j are not in serial position, we check for the conflicts. Lines 4 – 6 check for write-write conflicts from validating transaction T_j to conflicting transaction T_i. Write-write conflicts are allowed in the validation phase iff transaction T_i commits before transaction T_j starts validation phase. These conflicts are non-serious because in optimistic concurrency control protocol modifications to data item actually take place during the write phase. As write phase of transaction T_j has not yet started, these writes simply express the serialization constraint $T_i \rightarrow T_j$. Thus we can ignore these edges for cycle checking in the serialization graph.

Lines 7 – 14 check for read-write conflicts from validating transaction T_j to conflicting transaction T_i. Read-write conflicts are allowed in the validation phase iff all the elements in the ReadSet have timestamp larger than $T_i(ET)$, i.e., transaction T_j reads all conflicting data items, which were updated by transaction T_i, after T_i commits. As transaction T_i has already committed before start reading the conflicting data items by transaction T_j, these data items simply express the serialization constraint $T_i \rightarrow T_j$. Thus we can ignore these edges for cycle checking in the serialization graph.

Now if there remains write-read conflicts from validating transaction T_j to conflicting transaction T_i these conflicts will not be counted as we know actual writes take place only in write phase and T_j is now in validation phase, which precedes write phase. The validating transaction will be verified if both validating and conflicting transactions have common data items in the Read Sets (for this no edge will be appeared in the serialization graph). So there will be no any cycle in the serialization graph according to the algorithm. Thus timestamp based optimistic concurrency control protocol exhibits serializability.

VI. COMPARISON AND DISCUSSION

Here we are presenting a number of examples where original optimistic concurrency control protocol and timestamp based concurrency control protocol abort the validating transaction although it is not required. Instead our algorithm does not abort the validating transactions.

Consider the following Fig. 4 where two transactions T1 and T2 are accessing same data item x. Although timestamp based concurrency permits this scheduling, OCC does not permit the schedule and aborts the validating transaction. When T2 starts its validation phase, according to the original OCCP, T2 will be aborted as there is a read-write conflict between two transactions for data item x. But we can see from the figure that transaction T1 has finished its write phase before T2 reads the value of x. So T2 reads the updated value and there is no problem in serializability unless there is another data which is read by T2 before it is updated by T1. That's why according to our algorithm T2 is validated.

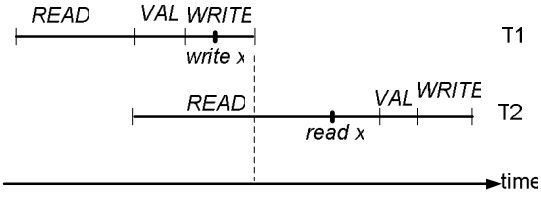

Fig. 4 Transaction T1 and T2

The following example shows a situation where both OCC and timestamp based protocols are not permitting a transaction to execute for completion where timestamp based OCC protocol is authorizing to do so. Consider the following Fig. 5 where transaction T1 and T2 are accessing data item x. According to the timestamp based concurrency control protocol, as T1 started later than T2, T1 has a large timestamp than T2. Whenever T2 wants to read data item x, it will be aborted by timestamp based concurrency control protocol since x has a write timestamp that was set by the timestamp of T1. Again, according to the OCC protocol transaction T2 will be aborted. This abortion of transaction T2 is unnecessary unless there is another data which is read by Ts before it is updated by T1. This situation may be very much common for database systems where some data items are stored in disks and some are in memory – clearly transactions accessing to memory will have a short read phase (like transaction T1 in Fig. 5) and transactions accessing to disks will have a comparatively longer read phase (like transaction T2 in Fig. 5).

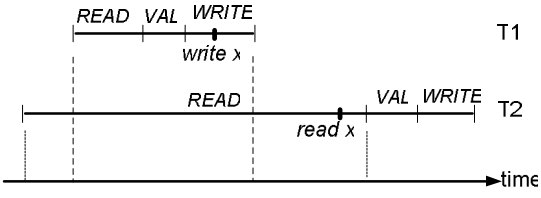

Fig. 5 Transaction T1 and T2

Timestamp based optimistic concurrency control protocol eliminates the risk of transaction being restarted in cases where detected conflicts actually do not endanger serializability. This protocol does not show any bias to long or short transaction. Moreover, the conflicts are specifically defined. The only overhead of our protocol is to maintain the timestamps whenever a transaction access data items. In timestamp based concurrency control protocol this is done with extra other modification like comparing the timestamp of a data item with that of a transaction, updating timestamp every time a data item is accessed etc. Also in optimistic concurrency control time marker is maintained. From these considerations, our algorithm adds minimal overhead to timestamp based and optimistic algorithms and is negligible compared to the cost where a transaction is unnecessarily aborted and then restarted.

VII. CONCLUSION AND FUTURE WORK

In this paper we have presented a timestamp based optimistic concurrency control protocol. We have tried to reduce the chances where original optimistic concurrency control protocol identifies conflicts that can be ignored. Thus obviously our protocol shows better performance.

There are still several issues to research

1. If the two transaction conflicts with each other which transaction has to be aborted – the validating transaction or the conflicting transaction. There may be some prospects if we think about transaction priority here.

2. After finding a validating transaction conflicting with another transaction how much time should have to wait to restart the aborted transaction. If it is restarted very soon there remains probability to conflict again. On the other hand if the transaction is restarted after some period of time the aborted transaction, especially if is it a real time one, may fail to meet its deadline.

3. Obviously if the conflicting ratio among the transactions is very high, optimistic concurrency control will not show a fine performance. If the conflicting ratio is high, lock based protocol is a good choice for concurrency control. So it will be a good research if it can be figured out the threshold point in this case.

REFERENCES

[1] Bernstein, P., Hadzilacos, V. and Goodman, N. (1987) *Concurrency Control and Recovery in Database Systems*. Addison Wesley, Reading, MA.

[2] Maurice Herlihy, *Apologizing Versus Asking Permission: Optimistic Concurrency Control for Abstract Data Types* ACM Transactions on Database Systems, Vol. 15, No. 1, March 1990, Pages 96-124

[3] Tanenbaum, A. S., *Distributed Operating Systems,* Pearson Education (Singapore) Pte. Ltd. (2002)

[4] Lunch, Nancy A., *Distributed Algorithms,* Morgan Kaufmann (1996).

[5] Sinha, P. K., *Distributed Operating Systems Concepts and Design*, Prentice-Hall of India Pvt. Ltd. (March 2002)

[6] Kung H. T., Robinson J. T., *On Optimistic Methods for Concurrency Control,* ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, pp 213-226.

[7] Stearns, R. E., Lewis, P. M., II, and Rosenkrantz, D. J. *Concurrency control for database systems*. In Proc. 7th Symp. Foundations of Computer Science, 1976, pp. 19-32.

[8] Jan Lindstr¨om, Seminar on Real-time Systems, University of Helsinki, Department of Computer Science, 1997

[9] Jan Lindstr¨om, *Optimistic Concurrency Control Methods for Real-Time Database Systems*, University of Helsinki Department of Computer Science Series of Publications C Report C-2001-9

[10] Agraval, R.: Concurrency Control and Recovery in Multiprocessor Database Machines: Design and Performance EvaluationP; Ph.D. thesis; University of Wisconsin, Madison; 1983

[11] Agraval, R., Carey, M., Livny M.: Models for Studying Concurrency Control Performance: Alternatives and Implications; Proc. ACM-Sigmod, International Conference on Management of Data; Austin, Texas; 1985

[12] Agraval, R., Carey, M., Livny M.: Concurrency Control Performance Modeling: Alternatives and Implications; ACM Transactions on Database Systems; Vol. 12, No. 4; Dec. 1987

[13] Augustin, R., Prädel, U., Scholten, H.: Performance Analysis of Concurrency Control Algorithm in Database Systems: a Survey (in German); Research-Report No. 174; Department of Computer Science, University of Dortmund, Germany; 1984

[14] Bhargava, B.: An Optimistic Concurrency Control Algorithm and its Performance Evaluation against Locking Algorithms; Report of the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260 USA; 1980

[15] Bhargava, B.: Performance Evaluation of the Optimistic Approach to Distributed Database Systems and its Comparison to Locking; Report of the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260 USA, 1982

[16] Carey, M.: Modeling and Evaluation of Database Concurrency Control Algorithms; Ph.D. thesis; University of California, Berkeley; 1983

[17] Franaszek, P; Robinson, J. T.: Limitations of Concurrency in Transaction Processing; ACM Transactions on Database Systems; Vol. 10, No. 1; Mar. 1985

[18] Huang, J.; Stankovic, J.; Ramamrithan, K.; Towsley, D.: Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes; Proc. 17th Int. Conf. on Very Large Databases (VLDB); Barcelona, Spain; 1991

[19] Huang, J.; Stankovic, J.: Concurrency Control in Real-Time Database Systems: Optimistic Scheme vs Two-Phase Locking; Technical Report, COINS 90-121; University of Massachusetts; Nov. 1990

[20] Kersten, M., Tebra, H.: Application of an Optimistic Concurrency Control Method; SOFTWARE- Practice and experience, Vol. 14, Feb 1984

[21] Menasce, D., Nakanishi, T.: Optimistic Versus Pessimistic Concurrency Control Mechanisms in Database Management Systems; Information Systems, Vol. 7, No. 1, 1982

[22] Peinl, P., Reuter, A.: Empirical Comparison of Database Concurrency Control Schemes; Proc. of the 9th Int. Conf. on Very Large Data Bases (VLDB); Florence, Italy; 1983

[23] Tay Y.; Goodman, N.; Suri, R.: Performance Evaluation of Locking in Databases: A Survey; Technical Report; TR-17-84, Harvard Aikon Lab.; Cambridge, Mass.; 1984

.