International Journal of Engineering Research and General Science Volume 2, Issue 6, October-November, 2014
ISSN 2091-2730

Analysis of Scheduling Nested Transactions in Distributed Real-Time Environment

Anup A. Dange, Prof. Neha Khatri-Valmik
Department of Computer Science and Engineering
Everest Education Society's Group of Institutions
College of Engineering & Technology
Ohar, Aurangabad, Maharashtra, INDIA
dange.anup88@gmail.com
nehavalmik@gmail.com

Abstract— A lot of research work has been done in the field of real-time database system to seek optimizing transaction scheduling to ensure global serializability. Nested transaction offers more decomposable execution units and finer-grained control over concurrency and recovery than "flat" transactions. For most applications we believe that it is desirable to maintain database constancy. It is possible to maintain consistency without serializable schedules but this requires more specific information about the kinds of transactions being executed. Since we have assumed very little knowledge about transactions, serializability is the best way to achieve consistency.

Keywords— Serializability, Concurrency Control, Lock Mechanism, hierarchical and flat protocol commit protocol, Two Phase Locking, Transaction Optimization, priority assignment

1. Introduction

Today's Database Management Systems (DBMSs) work in multiuse environment where users access the database concurrently. Therefore the DBMSs control concurrent execution of user transactions, so its overall correction of the database is maintained. Transaction is a user program which accesses the database. Database concurrency control permits users to access a database in a multiprogrammed fashion while preserving the illusion that each user is executing alone on a dedicated system. A distributed database system allows applications to access data from local and remote databases. Distributed applications spread data over multiple databases on n number of machines. Several smaller sized servers can be less expensive and more flexible than one large size, centrally located server. Distributed configurations take advantage of small sized, powerful server machines and less expensive connectivity as an option. Distributed system allows you to store data at several sites and each site can transparently access all the data. The key goals of distributed database system are to maintain availability, accuracy, concurrency and recoverability.

multiple users are accessing the same database simultaneously, their data operations must have to be coordinated so that inconvenient result gets avoided and concurrency of the shared data is reserved. This is called concurrency control and should provide each concurrent user with the illusion that he is referencing independent, dedicated database. For concurrent transactions execution serializability is the major correctness criteria. It is considered the highest level of isolation between transactions and plays an essential role in concurrency control. Two-phase locking is the most common method for concurrency control among transactions. Also this has been accepted as standard solution.

1.1 Commit Protocols

Enough research has been done on commit processing of flat transactions. The protocols are one phase commit, two phase commit and three phase commit protocols, PROPT real-time commit protocol and many others according to their optimizations. These protocols involve in transferring many messages in any of phase between participants where distributed transactions are executed. During this process many log records are generated and some are forced to dumped or flushed into the disk immediately in a synchronous manner. Due to this logging and messaging, commit protocols significantly increase the execution time for transaction execution. This creates problem to meet need of real-time context, this ultimately results in violation of timing constraints imposed on transactions. Therefore, selection of commit protocol is an important design decision for DRTDBS. There are lots of papers those already discussed about this issue given solution as relaxation in traditional ways or notations of atomicity or strict resource allocations and performance guarantees from the system. These all according to Harista who proposed PROPT real-time commit protocol.

Nested transactions means hierarchy of transactions and sub transactions and other sub transaction within previous sub transaction or contain any of database operations (read and write). In whole nested transaction is a collection of sub transactions and atomic database operations that comprise whole atomic execution unit.

In this paper we focus on How to achieve global serializability through concurrency control and transaction commit protocols. The concurrency control mechanism can be thought of as a policy for resolving conflicts between two or more transactions that want to lock the same data object. For concurrency control we used lock mechanism, called 2PL-NT-HP to solve conflict problems between nested transactions. For nested real-time transactions we used hierarchical and flat protocols, called 2PC-RT-NT.

1.2 Organization

The remainder of this paper consists of: Next section describes existing nested transaction system model along with its characteristics. Section 3 contains a real-time scheduler with priority assignment and concurrency control for nested transactions named as Two Phase Commit Nested Transaction Hierarchical Protocol. 4th section contains study of traditional hierarchical and flat two phase commit protocol along with detailed information. Last Section consists of conclusion and future research direction.

2. Real Time Nested Transaction Model

Two main types of nested transaction models are (1) closed nested transaction and (2) open nested transaction. In the closed nested transaction, a sub transaction's effect not appears outside of its parents view. Here commitment of sub transaction is depends upon the commitment of parent, while in open nested transaction model, sub transaction execute and commit itself independently. Due to some semantics of transactions we just include closed nested transactions.

In nested transaction model sub transactions are appearing atomic to the surrounded transactions and may commit independently. Until and unless all child transactions are committed a transaction is not allowed to commit. If child aborts its parent transaction need not to abort instead it just performs its own task or recovery. This is important to achieve its goal transaction need to perform any of following task: (1) to ignore condition (2) to restart the sub transaction (3) to initiate new sub transactions.

Some of the characteristics of nested transactions are listed in following table.

Parameter	**Meanings**
Ti	A transaction/ sub transaction
D(Ti)	Deadline of Ti
P(Ti)	Priority of Ti
ArrTime(Ti)	Arrival time of Ti
SlackTime(Ti)	SlackTime of Ti
Starttime(Ti)	StartTime of Ti
ResTime(Ti)	Resource time that the transaction requires for its execution
RemExTime(Ti)	Remaining execution time of Ti
ElaExTime(Ti)	Elapsed execution time of Ti

Table1. Characteristics of Nested Transaction

3. Scheduling Real-Time Nested Transactions

3.1 Priority Assignment Policy

Different types of priority assignment policies are there for flat transactions in Real time database management system. Without priority both transactions Ti and Tj shares CPU and disk units. Due to this any of transaction misses its deadline before completion of others work. Due to this best policy that is EearliestDeadLineFirst (EDF) priority assignment is the best in terms of success ratio. EDF assigns priority on the base of deadline. It assigns highest priority to earliest deadlined transaction. The formula for priority is given by:

$P(Ti) = 1/D(Ti)$

3.2 Sub transaction Priority Assignment

Along with main transaction priority there is need to prioritize subtransactions also. This help to ensure that transaction is not get delayed due to data conflict. There is need to assign deadline to subtransactions also. This deadline is according to transaction deadline and individual workload of subtransaction. But this does not improve success ratio as proven in [4]. There is another way to assign priority to transaction as describe in [5]. According to [5] addition of small p-value to the overall priority of transaction, this might help to prioritize subtransactions within a transaction such that this does not effect on priority assignment policy based on EDF. As child subtransaction must get complete before its parent subtransaction or transaction this is been done by assigning higher priority to a child subtransaction than parent transaction. This help to avoid intra-deadlock. Formula to calculate subtransaction priority is given by:

Subtransaction_priority = transaction_priority + subtransaction_level

Where, *subtransaction_level* is 0 for top level transaction. Nest to top level transaction has level value as 1, and so on, level value increases by 1 in each next level down in transaction tree.

4. Real-Time Concurrency Control

The most important characteristic of the cconcurrency control protocol is performance. In conventional database systems performance is usually measeured as the number of trasactions per second. In real-time databases performane is depends upon many other criteria, which are related to real-time. Some of these are the number of transactions that missed their deadlines, average tardy time etc. Due to new goals of optimization the algorithm that are used in the conventional daatabase systems do not show best result.

4.1 Data Conflict

Data Conflicts between committing and executing transactions are not uncommon compared with data conflicts between executing transactions. For example, in two phase locking protocol if an executing transaction request a data item which being locked by another transaction in a conflicting mode, the lock request will be denied and the executing transaction will be blocked until the lock is released.

4.2 Classical 2PL for Nested Transactions

Locking protocol provides two modes of synchronization [16]: Read: permits number of transactions' to access database at a time. Write: permits a single transaction for accessing a data item[12,9]. Transaction can acquire a lock on data items in any of the mode as M(Read,Write) and it holds until its termination. Other than holding a lock on data item a transaction can retain a lock when one of its subtransaction already committed. Its parent transaction inherits lock and then can retain them. There is a difference between holding and retaining lock; if a transaction holds a lock then it can access locked data items in given mode but this is not true for retaining a lock. Suppose write lock is retained by transaction Ti then subtransaction outside the hierarchy of retained lock is not able to acquire the lock. Whereas its descendent can acquire lock in read as well as write mode. This is not true in case of read lock. If a subtransaction Ti retains lock in write mode then any non descendent can holds the lock but only in read mode not in write mode and it remains lock retainer until its termination (i.e. commit or abort).

4.3 Real-Time 2PL for Nested Transactions

During concurrent execution of transactions it is important to maintain concurrency to order the updates in databases to maintain serializability. To do so in most of flat and nested transaction models locking data item is standard method. In real-time environment to maintain serializability a real-time concurrency control model was put forward by M. Abdouli, B. Sadeg and L.Amanton in their paper termed as 2PL-NT-HP[]. It used to solve data conflict problem occur between subtransactions by allowing transactions/subtransactions with higher priority to access data item and blocks or abort lower priority transaction. This 2PL-NT-HP is extended model of classical 2PL-NT, where some extra characteristics are added to it. These characteristics are combination of priority inheritance, priority abort, conditional restart and controller of wasted resources. Other than these there are some more characteristics of 2PL-NT-HP.

4.3.1 Priority Inversion

When the priority driven preemptive scheduling approach and two-phase locking protocol are simply inherited together, a problem occurs called as priority inversion. This occurs when higher priority transaction has to wait for execution of lower priority (sub) transaction.

For example, suppose transaction T_H has highest priority and T_L has lower priority than T_H and T_H is blocked by T_L due to access conflict. In this case T_H waiting for lock and T_L is executing, some transaction T_M may arrive, whose priority lies in between priorities of T_H and T_L then it just preempt T_L and take over the CPU even though T_M has no data conflict with T_L or T_H. This eventually delays the execution of T_H.

4.3.2 Priority Inheritance

Under this, when priority inversion occurs low priority transaction holding lock will execute at the priority of highest priority transaction waiting for the lock, until it terminates (abort or commit). In this way the lock acquiring transaction executes faster and releases lock quickly. This results in reduction of blocking time of higher priority transaction.

For example, suppose T_H is blocked by T_L due to data access conflict. Then, by using priority inheritance, T_L will execute at the priority of T_H. Now if T_M, an inheritance priority transaction, arrives, it cannot preempt T_L since its priority is less than the inherited priority of T_L. Thus T_H will not be delayed by T_M.

4.3.3 Priority Abort

Priority inversion problem is overcome by using priority abort scheme. In this lower priority transaction is aborted when higher priority transaction requires lock which is hold by lower priority transaction.

For example, when transaction T_H conflicts with a lock holding transaction T_L, T_L is aborted if T_H's priority is higher than that of T_L. Otherwise, T_H will wait for T_L. In this way, a high priority transaction will never be blocked by any lower priority transactions. Therefore priority inversion is completely eliminated.

4.3.4 Conditional Restart and Controller of Waste Resources

Conditional restart is employed to avoid the starvation problem come across during scheduling by using EDF, the mechanism is as follow: If slack time of (sub) transaction with higher priority is enough to execute after all lower priority (sub) transactions and then allowing these lower priority (sub) transactions to access data items firsts instead of aborting lower priority (sub) transactions. In other case these lower priority (sub) transactions must be aborted and restarted if controller of water resources allows it.

The controller of waste resources is a mechanism that checks if the restarted (sub) transaction can achieve its work before its deadline, using this mechanism, we reduce the wasting of resources as well as time.

5. Distributed Real-Time Nested Transaction Commit Protocol

For the nested transaction model a variety of protocols are already developed most are based on classical 2PC protocol. In our paper we focus on distributed real-time 2PC for nested transactions. But before directly working with 2PC we first study classical 2PC protocol.

5.1The Classical 2PC protocol

In the classical 2PC protocol, there are coordinator and participants. There is no direct communication between coordinator and participants apart from participants inform the coordinator while they join the transaction. Coordinator and participants communicate using message passing technique. Client can directly request to coordinator to commit (or abort) transaction. If client request *aborttransaction* the transaction then coordinator informs this to every participant immediately. If client request for commit transaction then 2PL comes to existence. In the first phase of the two-phase commit protocol the coordinator asks all the participants if they are prepared to commit this is called 'voting phase'; in the second, it tells them to commit (or abort) the transaction; this is called 'decision phase'. This is illustrated in fig.

Phase 1 (voting phase):
1. The coordinator sends a *canCommit?* request to each of the participants in the transaction.
2. When a participant receives a *canCommit?* request it replies with its vote (*Yes* or *No*) to the coordinator. Before voting *Yes*, it prepares to commit by saving objects in permanent storage. If the vote is *No*, the participant aborts immediately.

Phase 2 (completion according to outcome of vote):
3. The coordinator collects the votes (including its own).
(a) If there are no failures and all the votes are *Yes*, the coordinator decides to commit the transaction and sends a *doCommit* request to each of the participants.
(b)Otherwise, the coordinator decides to abort the transaction and sends *doAbort* requests to all participants that voted *Yes*.
4. Participants that voted *Yes* are waiting for a *doCommit* or *doAbort* request from the coordinator. When a participant receives one of these messages it acts accordingly and, in the case of commit, makes a *haveCommitted* call as confirmation to the coordinator.

Fig. 1 Two Phase Commit Protocol

This above protocol invokes only after receiving the *WORKDONE* message from the participant to the coordinator. This indicates transaction is ready to commit that is the work assigned to it is been completed. After receiving *WORKDONE* message coordinator broadcast *PREPARE* message to the participants. Those participants are ready to commit are replied as *YES* vote and acquires a prepared state. This prepared state means not unilaterally abort or commit the transaction alone. It has to wait for the final decision from the coordinator. On the other hand those participants wants to abort sends *NO* vote; it acts as veto to participants. This result in unilateral abort of transaction without waiting for the final outcomes from the coordinator. After receiving vote from each participant coordinator starts second phase. If all votes are *YES* then coordinator commits transactions by sending message as *COMMIT* to each participant; but any of participant votes *NO* then transaction is aborted by sending *ABORT* message to all participants. In both cased *ACK* message receives to the coordinator and all the resources get released.

5.2Real-Time 2PC for nested transactions

The **nested 2PC protocol** (also called *Tree 2PC* or *Recursive 2PC*) is a common variant of 2PC in a computer network, which better utilizes the underlying communication infrastructure. The participants in a distributed transaction are typically invoked in an order which defines a tree structure, the *invocation tree*, where the participants are the nodes and the edges are the invocations (communication links). The same tree is commonly utilized to complete the transaction by a 2PC protocol, but also another communication tree can be utilized for this, in principle. In a tree 2PC the coordinator is considered the root ("top") of a communication tree (inverted tree), while the cohorts (participants) are the other nodes. The coordinator can be the node that originated the transaction (invoked recursively (transitively) the other participants), but also another node in the same tree can take the coordinator role instead. 2PC messages from the coordinator are propagated "down" the tree, while messages to the coordinator are "collected" by a cohort from all the cohorts below it, before it sends the appropriate message "up" the tree (except an **abort** message, which is propagated "up" immediately upon receiving it or if the current cohort initiates the abort).

ACKNOWLEDGMENT

There is always a sense of gratitude, which everyone expresses for others for the help that has been rendered a crucial point in life and which facilitated the achievements of goals. I want to express a deepest gratitude to everyone who has helped me in completing this Seminar report successfully.

I feel extremely honored for the opportunity to work under the guidance of **Prof. Neha Khatri-Valmik.** Her eagerness to discuss on the topic and offer suggestion has been constant of encouragement of this work. I express my indebtedness to her support.

I am also thankful to **Prof. R.A. Auti**, Head, Computer Science & Engineering Department, Everest Education Society's College Of Engineering and Technology, Aurangabad.

I am also thankful to **Prof. Vankatesh Gaddime**, Principal, Everest Education Society's College Of Engineering and Technology, Aurangabad; for providing all necessary facilities at the college level and many helpful suggestions.

CONCLUSION AND FUTURE WORK

In this paper we studied the performance of distributed real-time nested transactions. We have used comprehensive approach to ensure global serializability such as real-time concurrency control approach and to maintain atomicity we used real-time two phase commit protocol. 2PL-NT-HP is a combination of properties like priority inheritance, priority abort, a conditional restart and controller of waste resources. Hierarchical 2PC is better than flat 2PC. The level size affects performance of real-time nested transactions. As we increase level size its performance gets decrease this is because of communication and number of messages exchanged on each level.

For the future work we use other protocols such as 3PC and PROPT protocols to enhance the performance of real-time nested transactions.

REFERENCES:

M. Abdouli, B. Sadeg and L. Amanton, "Scheduling Distributed Real-Time Nested Transactions," Eight IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05).

Hong-Ren Chen, and Y.H.Chin, "An Efficient Real-Time Scheduler for Nested Transaction Models", In Proceedings of the Ninth International Conference on Parallel and Distributed Systems (ICPADS'02),2002, pp.335-340.

K.Y. Lam, T.W. Kuo, and W.H. Tsang, "Concurrency Control in Mobile Distributed Real-Time Database Systems", Information Systems, 2000, vol.25, no.4, pp.261-286.

S.K. Lee, M. Kitsuregawa, and C.S. Hwang, "Efficient Processing of Wireless Read-Only Transactions in Data Broadcast", In Proceedings of 12th International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems RIDE-2EC, 2002,pp.101-111.

V.C.S. Lee, K.W. Lam, and S.H. Son, "On transaction Processing with Partial Validation and Timestamp Ordering in Mobile Broadcast Environments", IEEE Transactions on Computers, 2002, vol.51, no. 10, pp.1196-1211.

Lei Xiangdong, Zhao Yuelon, and Yuan Xiaol, "Transaction Processing in Mobile Database Systems", Chinese Journal of Electronics, 2005, vol.14, no.3, pp.491-494.

V.C.S. Lee, K.W. Lam, and T.W. Kuo, "Efficient Validation of Mobile Transactions in Wireless Environments", The journal of Systems and Software, 2004, vol.69, no.1, pp.l83-193.

Hong-Ren Chen, and Y.H. Chin, "Scheduling Value-Based Nested Transactions in Distributed Real-Time Database Systems", Real-Time Systems, 2004, vol.27, pp.237-269.

M. Abdouli, B. Sadeg, and L. Amanto, "Scheduling Distributed Real-Time Nested Transactions", In Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05), 2005, pp.208-215.

Liao Guoqiong, Liu Yungsheng, and Wang Lina, "Concurrency Control of Real-Time Transactions with Disconnections in Mobile Computing Environment", In Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing (ICCNMC'03), 2003, pp.205-212.

G. Weikum, and G. Vossen. Transactional information system: theory, algorithms, and the practice of concurrency control and recovery, USA:Elsevier Science, 2003.

A. Data, and S.H. Son, "Limitations of Priority Cognizance in Conflict Resolution for Firm Real-time Database Systems", IEEE Transactions on Computers, 2000,vol.49, no.5, pp.483-501.

E. Pitoura, and P.K. Chrysanthis, "Multiversion Data Broadcast", IEEE Transactions on Computers, 2002, vol. 5, no. 10, pp.1224 - 1230.

J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

I.S. Jacobs and C.P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989