01 Jan 1997

User Defined Prewrites for Increasing Concurrency in Design Databases

Sanjay Kumar Madria
Missouri University of Science and Technology, madrias@mst.edu

A. Embong

Recommended Citation

S. K. Madria and A. Embong, "User Defined Prewrites for Increasing Concurrency in Design Databases," *Proceedings of 1997 International Conference on Information, Communications and Signal Processing, 1997*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1997.
The definitive version is available at https://doi.org/10.1109/ICICS.1997.652091

International Conference on
Information, Communications and Signal Processing
ICICS '97
Singapore, 9-12 September 1997

User Defined Prewrites for Increasing Concurrency in Design Databases

Sanjay Kumar Madria and Abdullah Embong

School of Computer Sciences
University Sains Malaysia
11800 Minden, Penang, Malaysia
skm@cs.usm.my, ae@cs.usm.my

Abstract

In this paper, we introduce a prewrite operation before a write operation is performed on design databases, a database that consists of objects as engineering designs. A prewrite operation does not actually make a design but it only makes available the model of the design that the transaction will produce in future. Once the prewrite design by a transaction is announced, the transaction executes a pre-commit operation. After the pre-commit, read operations can access the prewrite design even before the pre-committed transaction has finally made the design and committed. Therefore, our algorithm increases the potential concurrency as compared to the algorithms that permit only read and write operations on the design objects. In our algorithm, a user explicitly makes available a prewrite model of the design to be finally produced. Similarly, a read transaction also mentions whether it wants to read a prewrite version or the final version of the design. Locking protocols using two phase locking are given to control concurrent operations.

Keywords : Prewrite, design database, pre-commit, concurrency.

1. Introduction

In a database system, users make access to database by executing read and write operations. A read operation on a database object does not conflict with another read operation since neither of them modifies the data object. A write operation on the otherhand conflicts with both read and write operations. A concurrency control algorithm [1,2,3,4] is needed to control interleaving of conflicting operations that otherwise violate the consistency of the database.

Many concurrency control protocols are based on the notion of locks [1,2,5,6,7] where a data object in the database can be accessed only after a lock on that object has been acquired for the duration of the read or writes. That is, a read (write) operation can be executed only after a read(write)-lock is obtained. After acquiring the locks, the transaction executes its operations and then may release the locks. Since read operations do not conflict, many read transactions may share the read-lock on an object but sharing is not permitted if one of the lock is a write-lock. The above model is more suitable for database management systems that support short-duration transactions that read and write data objects for a short period of time. However, the long-duration transactions [8,9,10,11] which mainly occurs in engineering design [11] or large software development applications [10], access large volume of data objects. Therefore, in such cases, the traditional concurrency control algorithms based on above protocols suffer from performance degradation. Due to isolation requirements of these protocols, the designs can not be released for viewing until the transaction commits. Therefore, once the transaction acquires a write-lock on the design, the other transactions have to wait. Thus, if a short-duration transaction wants to access the design objects held by a long transaction, it will end up waiting for the long-lived transaction to commit. Also, if a transaction is considered as a unit of work, a significant amount of work may be lost in case of a failure. A read transaction should not wait for very long in order to access the database designs. Therefore, it is desirable to make the response of a system fast for read-only transactions. Furthermore, the system should not delay short-duration transactions due to the presence of long transactions.

The transaction models developed for early database applications fall short of meeting

0-7803-3676-3/97/$10.00

the requirements of these new transactions has also been shown in [12,13]. Also, some of the proposed algorithms [14,15] to manage long-duration transactions do not rely on serializability [1,2] and therefore, uses different correctness criterion.

In this paper, we present a new transaction model using prewrite operations [16,17,18] to increase concurrency during transaction processing in design databases.

2. Overview of Our Transaction Model

We use a prewrite operation [16,17,18] before an actual write operation is executed on design objects to increase the potential concurrency. A prewrite operation makes available the model of the design that the object will have after the design is finally produced. A prewrite operation does not make the design but only provides the model or the picture of the design (including its dimensions, colour combination etc.) a transaction intends to make in future. Once the prewrite design of a transaction is announced, the associated transaction executes a pre-commit operation. After the transaction has finally produced the design (for which the prewrite design has been announced), it commits. A read transaction can read the prewrite design before the pre-committed transaction has produced that design. The prewrite design is made available for reads after the associated transaction has executed a pre-commit but before it has been committed. Hence, prewrites increase concurrency as compared to the environment where only read and write operations are allowed on design objects.

Once a transaction has announced a pre-commit, it is not allowed to abort. This is due to the following reasons. First, it will help in avoiding cascading aborts [1] since the prewrite design has been made available before the transaction has finally produced the design and committed. Second, it is desirable for a long-duration transaction so that it does not lose all the design work at finishing stage in case there is an abort or a system failure. To accomplish this, a pre-commit operation is executed only after all the prewrite log records are stored on stable storage. Once write operations start, each write log is also stored on stable storage. Thus, in case of a failure after pre-commit, there is no need of executing rollback (undo) operations. The recovery algorithm has to at the most redo those operations (using the prewrite logs and the write logs) whose effects are not there on stable storage [1,16,17]. The failed transaction then can restart from the state as exists at the time of failure [16]. If a transaction aborts before executing all prewrite operations and a pre-commit, it is rolled back by discarding all the announced prewrites.

In engineering design applications [8,19], by introducing prewrite designs, short-duration transactions can access the model or picture or the working copy of the design held by a long transaction. Reads are allowed to access the sketches once they are prewritten but before they are actually made. Therefore, using prewrites, one can have a system consisting of short and long transactions without causing delay for short-duration transactions. Thus, prewrites help in increasing the throughput of the system by making the response of the system faster for read-only operations. For read-only transactions, the picture or the model of the design to be produced is important rather then the finally completed design. In the final production, the dimension or certain colour combinations of the design may vary from the prewrite version of the design, however, the read-only transactions are not affected.

In our algorithm, the user transactions explicitly mention a prewrite operation before an actual write. Also, user transactions explicitly mention whether it wants to access the prewrite design (we call it pre-read operation) or the final design (we call it read operation). That is, the existence of a prewrite is visible to the scheduler, data manager (DM) and to user transactions. The user transaction submits prewrite, pre-read, write and read for the design objects it wants to access. Once a transaction is submitted to the Data Manager (DM), the DM analyses the received transaction. If the transaction has a prewrite operation, the DM will store the announced design for the object in the prewrite-buffer. If the transaction has a pre-read operation, the DM will return the corresponding prewrite design from the prewrite-buffer. If the operation is a read, it returns the final produced design from the write-buffer.

In our algorithm, two versions of the same design may be available for reading. The first version is the final design released for manufacturing or the last design checked for correctness. The other is the most recent working copy of the design (prewrite design). However, after the final version of the design is produced, prewrite version of the design will be no longer available. That is, after the final design is released, a read transaction can not access its prewrite design. Also, the independent writing on these two different versions of design are also

not allowed. Therefore, our algorithm is different from the multiversion algorithm [1]. In the multi-version algorithm, versions are not visible to users and versions can be read and written independently.

We have used two phase locking algorithm to control concurrent operations.

3. Concurrency Control Algorithm

In our concurrency control algorithm, we assume that the users explicitly mention prewrite designs in their transactions. Also, a read transaction explicitly specifies whether it wants to access the prewrite design or the final design. That is, whether the read transaction wants to access the working copy of the design to be completed and produced later or it only wants to access the final design to be released for production. After the prewrite design is made, a transaction executes a pre-commit operation. After the corresponding design is finally produced, a transaction commits.

In order to design a concurrency control protocol, we first analyse the conflicting and non-conflicting operations in our model. The following operations on the same design object conflict :

1. Two prewrites conflict since two prewrite design for the same object can not be announced at the same time in the same prewrite-buffer. It produces a prewrite-prewrite type of the conflict.
2. A prewrite operation conflicts with a pre-read operation since pre-read returns the value from the prewrite-buffer whereas prewrite changes the contents of the prewrite-buffer. Therefore, it will generate a conflict of the type pre-read and prewrite.
3. Two writes conflict since both of them will modify the same design object at the same time. This will generate a write-write type of the conflict.
4. A write operation conflicts with a read operation since the read returns the value from the write-buffer. and a write changes its contents. Therefore, it will generate a conflict of the type read and write.

The following operations, in general, do not conflict :

1. A pre-read operation (to read a prewrite design) and a write operation do not conflict. A write operation operates on the write-buffer whereas a pre-read operates on the prewrite-buffer.
2. A prewrite and write conflict do not conflict as prewrite and writes operate on their respective buffers.
3. A read and prewrite do not conflict as they operate on different buffers; read operates on write-buffer whereas prewrite operates on the prewrite-buffer.

The following is the conflict detection matrix :

	pre-read	read	prewrite	write
pre-read	no	no	yes	no
read	no	no	no	yes
prewrite	yes	no	yes	no
write	no	yes	no	yes

4. Locking Protocols

Our concurrency control protocols are based on two phase locking [1,2]. The protocols based on locking delay the execution of conflicting operations by using read-locks for read and pre-read operations, prewrite-locks and write-locks for prewrite and write operations, respectively. Two phase locking requires a transaction to acquire all locks before releasing any locks. We have the following locking rules in order to control the concurrent operations such that two conflicting operations should not get their locks at the same time.

1. If a prewrite-lock is held by a transaction, no other transaction can acquire a prewrite-lock or read-lock for pre-read (to read the prewrite value).
2. If a transaction acquired a write-lock, no other transaction can acquire a write-lock, or a read-lock for a read operation (to read the final design).
3. When a transaction holding a read-lock (either to read a prewrite or write value), other transactions can hold read-locks.
4. Once a transaction announces a pre-commit, all its prewrite-locks are converted to write-locks.
5. If a transaction is holding a prewrite-lock, other transaction can hold a write-lock as they operates on different buffers. However, since a prewrite-lock is never released but it is converted to a write-lock. Therefore, it may result in a write-write conflict. If we have used a prewrite-lock and write-lock separately, we can not release a prewrite-lock before acquiring a write-lock because of two phase locking condition.

To deal with such a situation, we propose two solutions.

- A prewrite-lock is updated to a write-lock provided no conflicting locks are held by other trannsactions on the corresponding data objects. Otherwise, transactions have to

wait in an ordered queue to convert their prewrite-locks to write-locks.

- To use ordered-shared locking [20] with some modifications. We allow prewrite and write operations of the two transactions T_1 and T_2 to operate concurrently by acquiring respective locks in an ordered fashion. That is, if a write-lock is acquired by a transaction T_1 then a prewrite-lock can be acquired by transaction T_2. We allow the prewrite operation to execute before the write operation. This is an improvement over ordered-shared locking [20] where the operations must execute in the order the ordered-shared locks are acquired. Once the prewrite-lock of T_2 is converted to the write-lock, the two write operations of the transactions are executed in the order their write-locks are acquired. Another improvement over [20] is that transaction T_2 can commit before T_1. In [20], the transactions with ordered-shared locking are allowed to commit in the order they obtained their locks.

6. When a transaction is committed or aborted, all its locks are released.

5. Conclusion

In this paper, we have presented a transaction model using prewrite operations to increase concurrency in design database environment. Our transaction model provides more concurrency as model of the designs are made available to read-only transactions before the designs are finally produced. We have designed the locking protocol for controlling the concurrent conflicting operations in our model. Our model provides no undo type of recovery [16] in case of transaction aborts. We are studying this model in mobile database and work-flow environments.

References

[1] P.A. Bernstein, V. Hadzilacose and N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison-Wesley, Reading, MA,1987.

[2] C.H. Papadimitriou, "The Theory of Database Concurrency Control", Computer Science Press, Rockville, MD, 1986.

[3] W.E. Weihl, "Commutativity-based Concurrency Control for Abstract Data Types", IEEE Transactions on Computers", 37, No.12, pp. 1488-1505, Dec.1988.

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

[5] M. Yannakakis, "Serializability by Locking", Journal of ACM, 31, No.2, pp. 227-244, 1984.

[6] K.P. Eswaran, J.N.Gray, R.A. Lorie and I.L. Traiger, "The Notion of Consistency and Predicate Locks in Database Systems", Communication of ACM, 19, No.11, pp. 624-633, 1976.

[7] H.F. Korth, "Locking Primitives in Database Systems",Journal of ACM, 30, No.11, pp. 55-79, 1983.

[8] F. Bancilhon, W. Kim, and H.F. Korth, "A Model of CAD Transactions", in Proceedings of the 11th International conference on Very Large Databases, VLDB Endowment, pp. 25-33, 1985.

[9] V. Dayal, M. Hsu, and R. Ladin, "Organizing Long-running Activities with Triggers and Transactions", in Proceedings of the ACM SIGMOD international Conference on Management of Data, ACM, New York, pp. 204-214, 1990.

[10] H.F. Korth and G. Speegle, "Long Duration Transactions in Software Design Projects", in 6th International Conference on Data Engineering, IEEE, New York, pp. 568-674, 1990.

[11] H.F. Korth, W. Kim, and F. Bancilhon, "On Long-Duration CAD Transactions", Information Science, 46, pp. 73-107, Oct.1990..

[12] C. Beeri, P.A. Bernstein, and N. Goodman, "A Model for Concurrency in Nested Transaction System", Journal of ACM, Vol.36, No.2, pp.230-269, 1989.

[13] G. Weikum, "Principles and Realization Strategies of Multi-level Transaction Management, ACM Transaction on Database Systems", Vol.16, No.1, pp.132-180, 1991.

[14] H.F. Korth, and G. Speegle, "Formal Model of Correctness without Serializability", In SIGMOD International Conference on Management of Data, ACM, NewYork, pp.379-386,1988.

[15] H. F., Korth, G. Speegle, " Formal Aspects of Concurrency Control in Long-duration Transaction Systems using NT/PV Model", ACM Transactions on Database Systems, Vol.19, No.3, pp.492-535, Sept.1994.

[16] S.K. Madria, "Concurrency Control and Recovery Algorithms in Nested Transaction Environment", Ph.D. Thesis, Indian Institute of Technology, Delhi, India, 1995.

[17] S.K. Madria, S.N. Maheshwari, B. Chandra, B. Bhargava, "Crash Recovery Algorithm in an Open and Safe Nested

Transaction Model", 8th International Conference on Database and Expert System Applications (DEXA'97), France, Sept.97, Lecture Notes in Computer Science, Springer Verlag, 1997.
[18] S. K. Madria, and B. Bhargava, "System Defined Prewrites to Increase Concurrency in Databases", First East-Europian Symposium on Advances in Databases and Information Systems (in co-operation with ACM-SIGMOD), St.-Petersburg (Russia), Sept.97.
[19]. W. Kim, R. Lorie, D. Mcnabb, and W. Plouffe, "A Transaction Mechanism for Engineering Design Databases", in Proceedings of the 10th International Conference on Very Large Databases, VLDB Endowment, pp. 355-362, 1984.
[20] Agrawal, D., and Abbadi, A., "A Non-restrictive Concurrency Control Protocol for Object Oriented Databases", Distributed and parallel Databases: An International Journal, Vol.2, No. 1, pp. 7-31, Jan.1994.