Journal of Korea Multimedia Society Vol. 7, No. 5, May 2004(pp. 633-639)

A Write Notification Approach for Optimistic Concurrency Control Schemes

SungChan Hong†

ABSTRACT

The performance of optimistic concurrency control schemes which are generally used for Mobile computing is very sensitive to the transaction abort rate. Even if the abort probability can be reduced by back-shifting the timestamp from the time of requesting a commit, some transactions continuously perform unnecessary operations after the transactions accessed write-write conflicting data. In this paper, we propose an optimistic protocol that can abort the transactions during the execution phase by using the write notification approach. The proposed protocol enhances the performance of the optimistic concurrency control by reducing the unnecessary operations. In addition, we present a simulation study that compares our schemes with the timestamp based certification scheme. This study shows that our scheme outperforms the timestamp based certification scheme.

낙관적 동시성 제어를 위한 쓰기 통지 기법

홍 성 찬†

요 약

일반적으로 모바일 컴퓨팅에 사용되는 낙관적 동시성 제어의 성능은 트랜잭션 철회율에 민감하다. 비록 완료를 요청한 시각의 타임스탬프를 뒤로 옮김으로 해서 철회 확률을 줄일 수 있지만 일부 트랜잭션은 쓰기-쓰기 충돌을 일으키는 데이터를 접근한 후에 계속적으로 필요 없는 연산을 수행한다. 본 논문에서는 쓰기 통지 접근방식을 이용하여 그러한 트랜잭션들을 실행단계에서 철회 시킬 수 있는 낙관적인 프로토콜을 제안한다. 제안하는 프로토콜은 필요없는 연산을 줄임으로서 낙관적 동시성 제어 기법의 성능을 향상시킨다. 또한, 타임스탬프를 기초로 한 프로토콜과 제안한 프로토콜과의 성능 비교를 제시한다. 성능평가에서 제안하는 방식이 타임스탬프를 이용하는 방식보다 높은 성능을 나타낸다는 것을 보인다.

Key words: Distributed System(분산 시스템), Transaction Processing(트랜잭션 처리), Concurrency Control(동시성 제어)

1. Introduction

The concurrency control schemes can profoundly affect the performance of transaction processing systems[1-6]. Generally, the concurrency control schemes can be characterized as either the optimistic approach or the pessimistic approach. In the optimistic concurrency control (OCC), transactions run without waiting to ensure non-conflicting access to data, and transactions are aborted if conflicting access is detected in the validation phase[5,10]. The OCC performs better than the pessimistic concurrency control method in an environment that has high message exchange costs or low contention[1,6]. However, the weak point of the OCC is that it is very sensitive to the

※ 교신저자(Corresponding Author) : 홍성찬, 주소 : 경기도 오산시 양산동(447-791), 전화 : 031)370-6796, FAX : 031)370-6984, E-mail : schong@hs.ac.kr
접수일 : 2003년 5월 12일, 완료일 : 2003년 11월 4일
† 한신대학교 정보통신학과 교수
※ 이 논문은 2003년도 한신대학교 학술연구비 지원에 의하여 연구되었음.

transaction abort rate.

Different methods of implementing the certification scheme for the OCC have been proposed. The popular variations of OCC are pure OCC[5], broadcast OCC[7], OCC based on timestamp history (TSH)[4], OCC with Serialization Graph[12, 13] and Distributed OCC[14]. In the Pure OCC, transactions are aborted only at the transaction commit time if conflicting access is detected. In the broadcast OCC[7], committing transactions cause the abort of conflicting transactions in the middle of their execution. Because OCC dose not use any locking mechanisms, many researches for the mobile computing suppose that validation based scheme will be more effective in the mobile computing environments[12,13]. However, the weak point of OCC with Serialization Graph[12,13] is that the space and time overhead in maintaining serialization graphs. In addition, the schemes have no mechanism reducing the abort rates.

The Distributed OCC[14] uses a broadcast OCC for first run and locking for second run. Even if the scheme reduces the abort rate by locking in second run, performance of the scheme is very similar with the standard locking approaches. TSH[4] can reduce the abort probability by back-shifting the timestamp from the time of requesting a commit. TSH outperforms the other OCC schemes because TSH only reduces the number of aborts without maintaining locking information, multi-version or serialization graphs.

Our primary goal is to enhance the performance of the TSH by reducing the unnecessary operations and certification overhead. We provide an algorithm to abort the transactions when it is accessing a write-write conflicting data item. In addition, we extend the algorithm to reduce the overhead of validation in this paper. Our scheme is an extension of the TSH and consists of additional data structures. Using our algorithm, transactions abort at an earlier time than the TSH without any spurious aborts.

2. Write Notification for Certification Protocol

In this paper, for the sake of notational simplicity T denotes timestamp, S denote set, R denote read, D denote data and W denote write. In the TSH, each transaction obtains a unique timestamp at the commit time. It maintains a read timestamp RT and write timestamp WT for each data item. The read timestamp and write timestamp are the timestamps of the most recent committed transaction that read the data and wrote the data, respectively. It maintains k write timestamps history (WT_1, ..., WT_k) for a data item, with WT_1 as the oldest update and WT_k as the lastest update, so that $WT = WT_k$. A transaction views each data item as a (*name*, *version*) pair. For each data read, the transaction tracks the WT of the data as its version. Before certification, a timestamp ST, equal to the certification time, is generated for the transaction. For each read operation RD_i, i = 1,..., m, for data D_i, the read version is checked with the current WT of the data item. If the read versions are equal to their current WTs, the transaction is certified and the read timestamps and the write timestamp history are updated. If it isn't, it simply means that those data items have been updated subsequently by other committed transactions. In this situation, the TSH tries to find an alternative back-shifted timestamp instead of the commit timestamp. For each RD_i, the TSH compares its version number accessed by the transaction with the write history $WT_1(RD_i)$, $WT_2(RD_i)$, ... , $WT_k(RD_i)$. If it can't find some $WT_j(RD_i)$ equal to the version read for any RD_i, the transaction is aborted. Otherwise, the valid interval of RD_i is ($WT_j(RD_i)$, $WT_{j+1}(RD_i)$) where $WT_j(RD_i)$ is equal to the version number accessed for RD_i. The valid interval of write timestamp $WT_j(RD_1)$ is ($WT_j(RD_1)$, ST). If there is no intersection of the valid intervals, the transaction is aborted. Additionally, if the upper bound of the

intersection of all intervals is equal to ST, no back-shift is needed. Otherwise, it will find a new timestamp NT. The TSH scheme checks whether its read timestamp, $RT(WD_i)$, is less than NT. If it fails to do this, the transaction is aborted. If the transaction has not yet been aborted by previous processing, it can be certified with the timestamp NT. For each WD_i, if NT is greater than $WT(WD_i)$, then an update is made to the database; otherwise, the write is not reflected in the database (*Thomas' write rule*[5]). In either case, the write history is updated accordingly. The read timestamp $RT(RD_i)$ is set to the maximum of the current RT and NT.

In TSH, some transactions continuously perform unnecessary operations even after they have accessed write-write conflicting data items, because they can not be aborted during execution phase. The difference between our approach and the TSH is that a transaction notifies the server whenever the transaction performs write operations. It works as follows. The server maintains a set of current running transaction identifiers CS. It also maintains a maximum timestamp MT_i for each transaction identifier T_i in set CS. Therefore, the set CS consists of (T_i, MT_i) pairs. Timestamp MT makes it possible to distinguish whether the transaction associated with timestamp MT accessed the conflicting data item or not. The initial value of timestamp MT is the initial value of the read timestamp (i.e., $MT = RT_i$). The server also maintains a set of read transactions(RS) and a set of write notifying transactions(WS) for each data item D. The set RS is maintained to update the timestamp MT and the set WS is maintained to decide whether to abort. When transaction T_i is started, the server inserts the transaction identifier T_i into set CS and sets timestamp MT_i of the transaction T_i to timestamp RT_I. Whenever transaction T_i reads each data item D_j, transaction identifier T_i is inserted into set RS_j. Whenever transaction T_i writes on each data item D_k, the transaction sends a write notifying message to the server. When the server gets the message, it checks whether transaction T_i has accessed any write-write conflicting data item or not. If it has accessed conflicting data items, then transaction T_i is aborted. However, if it has not accessed conflicting data items, then transaction identifier T_i is inserted into set WS_k.

When transaction T_i requests a commit, we use the validation algorithm in[4]. After transaction T_i passed the validation phase, we try to find conflicting transactions based on set RSs and WSs. For each data D_j that was read and written by transaction T_i, if a transaction identifier is in set RS_j and WS_j, the transaction is aborted. After all conflicting transactions have been aborted, all timestamp MT_I is updated only if transaction identifier T_I is in set RS_k of data D_k that was written by transaction T_i. Transaction T_i is committed after all information of transaction T_i have been deleted from set CSs, RSs and WSs.

Although clients do not send the write notification messages to the server, consistency is not broken. In our scheme, the write notification messages are used only to reduce unnecessary operations. Therefore, the write notification messages are piggy-backed on other messages being exchanged between the client and server. There is always a certain amount of such traffic. For example, clients send messages to the server for read operations or servers and clients exchange "*I'm alive*" messages for failure detection purposes. Therefore, our scheme does not cause any extra message traffic, although messages may be longer since write notification messages are piggy-backed in these messages.

When the server gets this message, it compares maximum timestamp MT_i with the current read timestamp RT_k on notified data item D_k. The transaction T_i is aborted only if the value of timestamp MT_i is not the initial value and timestamp MT_i is less than or equal to read

timestamp RT_k. This means that transaction T_i is accessing conflicting data, because timestamp MT_i is updated only if another committing transaction updates the data read by transaction T_i. In addition, if timestamp MT_i is less than or equal to RT_k, the transaction can not find a re-orderable timestamp. Therefore, the transaction is aborted and the algorithm is terminated. If transaction T_i is not aborted, transaction identifier T_i is inserted into set WS_k of notified data D_k.

Regardless of whether the transaction is committed normally or re-ordered, assume that the committing transaction identifier is T_j. For each accessed data D_k by committing transaction T_j, if transaction identifier T_j is in set WS_k and RS_k, and transaction identifier T_i is also in set WS_k and RS_k, then it means that transaction T_i read and wrote data D_k that was updated by transaction T_j. In this case, transaction T_i has to be aborted in the validation phase because it accessed write-write conflicting data D_k. Therefore, transaction T_i is aborted when transaction T_j is committed in our scheme. When transaction T_i is aborted, the information of transaction T_i is deleted from set CS.

When transaction T_j is committed normally with timestamp ST, each timestamp MT_l is set to timestamp ST only if timestamp $MT_l = RT_l$, where MT_l denotes the maximum timestamp of the transaction identifier T_l that is in set RS_l associated with data WD_k written by committing transaction T_j.

When transaction T_j is committed with a re-ordered timestamp OT, each timestamp MT_l will be set to a re-ordered timestamp OT only if timestamp $MT_l = RT_l$ or $MT_l > OT$, where MT_l denotes the maximum timestamp of transaction identifier T_l that is in set RS_k associated with data WD_k updated by re-ordered transaction T_j. Therefore, the value of maximum timestamp MT is never increased after the value of MT is changed from RT_l. After this happens, all transaction identifiers T_j in set RS_k or WS_k of accessed data item D_k by transactions T_j are deleted. After all transaction identifiers have been deleted, the information of transaction T_j is also deleted from the set CS.

To prove that our scheme is correct, we have to prove that all histories representing executions that could be produced by it are serializable. In TSH, the assignment of a transaction timestamp in the valid interval of all read data items to a committing transaction guarantees that the edges from the committing transaction are to transactions with a larger timestamp. Requiring the transaction timestamp to be larger than the read timestamp of updated data implies that no transaction with a larger timestamp has an edge into the committing transaction. This ensures that the serialization graph is acyclic. Because our certification scheme is based on TSH, all histories produced by our scheme are serializable by the Serializability Theorem.

3. Simulation Study

In this section, we present a simulation study to demonstrate that our Write Notification scheme (WN) outperforms the TSH. All the simulations were done using a discrete event model. Table 1 shows the relevant system parameters, settings and their meanings for our simulator. The parameters and settings were chosen based on[11]. In this study, the number of transactions is assumed to be parameter *Num_Tran* and the number of data is assumed tp be parameter *Num_Data*. The parameter OP denotes the mean number of operations accessed per transaction. The write probability is determined by the parameter *Per_Write*. Th parameters *Time_Read*, *Time_Write*, *Time_Net*, *Time_Restart* and *Time_Valid* denote processing time. In this study, we set 3 for write timestamp history.

Fig. 1 presents abort rates with various numbers of transactions and operations when the write probability is set to 20%. Fig. 1 shows that the

Table 1. Parameters, Settings and their meanings

Parameters	Settings	Meanings
Num_Data	1000	Number of data
Num_Tran	10~25	Number of transactions
OP	10, 20, 30	Number of operations
Per_Write	20%	Percentage of write probability
Time_Read	20 time units	Read access time
Time_Write	30 time units	Write access time
Time_Net	40 time units	Network delay
Time_Restart	20 time units	Re-start delay
Time_Valid	*Num_Oper* X *Time_Read*	Validation time
Hist_Size	3	Size of Write History

number of aborts of the WN is equal to or slightly greater than that of the TSH across a range of parameter setting. The reason of more aborts in the WN than the TSH is that some transactions can be aborted in their execution phase in our scheme. However, this case is rare. Additionally, even if the number of aborts is just little increased, our scheme reduces the unnecessary operations. This kind of aborts can be treated using the rerun policy introduced in[9].

Fig. 2 presents the number of re-ordered transactions with various numbers of transactions and operations when the write probability is set to 20%. The number of re-ordered transactions is a good evidence that shows our scheme does not make any spurious aborts. As Fig. 2 shows, the numbers of re-ordered transactions of the WN and

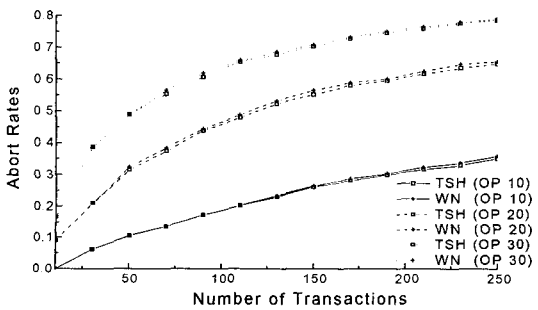

Fig. 1. Abort Rates (*Per_Write* 20%)

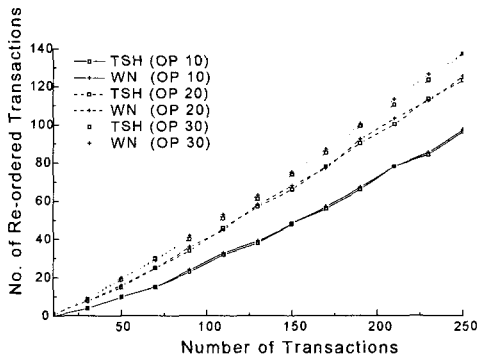

Fig. 2. Number of Re-ordered Transactions (*Per_Write* 20%)

the TSH are almost the same in all parameter setting. It means that the WN scheme does not make any spurious aborts that are occurred in the broadcast OCC. In other words, our scheme aborts only the transactions that would be aborted in the TSH scheme. The reason of the difference in the number of re-ordered transactions is that some transactions are aborted in the execution phase in our scheme. Hence, commit orders are not the same even if two schemes are simulated in the same environment. Changing commit order can effect the number of re-ordered transactions.

Fig. 3 presents the mean response time with various numbers of transactions and write probabilities when the number of operations is set to 10. A crucial point of the result is that the response time gap between the WN and the TSH grows as the write probability grows. Because our scheme aborts some transactions that accessed write-write conflicting data, the gap between the WN and the TSH becomes larger as the write probability increases.

Fig. 4 presents the mean response time with various numbers of transactions and write probabilities when the number of operations is set to 30. Changing the number of operations has large impact on the response time. It means that transaction length significantly affects the performance in the OCC schemes, because the abort probability of a long transaction is higher than that of a short

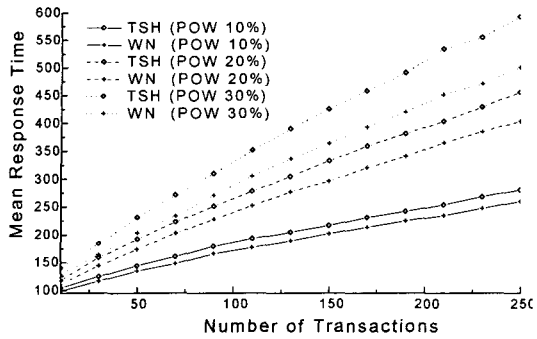

Fig. 3. Mean Response Time (*OP* 30)

one. Compared with Fig. 3, Fig. 4 shows more gaps between the WN and the TSH. Because the proposed scheme reduces the unnecessary operations, its benefit in terms of response time becomes larger as the number of operations increases.

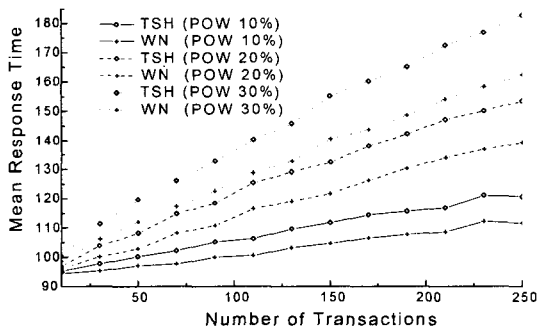

Fig. 4. Mean Response Time (*OP* 10)

4. Conclusion

In this paper, we proposed a protocol that can abort the transaction during the execution phase when it accesses a write-write conflicting data item using write notification approach. The difference between our approach and the certification based on timestamp history is that a transaction notifies its write operations to servers when the transaction performs write operations.

An important characteristic of our scheme is that our scheme does not make any spurious aborts that may happen in the broadcast optimistic concurrency control. Moreover, our scheme does not cause extra message traffic, because the write notification messages are piggy-backed on other messages. By the simulation, we showed that our scheme outperforms the certification based on timestamp history with reduction of unnecessary operations. In addition, performance gap between our scheme and optimistic concurrency control with timestamp history becomes larger as the number of operations and the write probability increase. Hence, our scheme is a suitable scheme in an environment that has long transactions and high write probability.

References

[1] R. K. Agrawal, M.J. Carey, and M. Livny, "Concurrency Control Performance Modeling: Alternatives and Implications," *ACM Trans. Database Syst.*, vol. 12, no. 4, pp. 609-654, Dec. 1987.

[2] M. A. Bassiouni and U. Khamare, "Optimistic Concurrency Control Schemes for Performance Enhancement," *in Proc. COMPSAC 86*, Chicago, IL, Oct. 1986, pp. 43-49.

[3] P. S. Yu and D. M. Dias, "Impact of Large Memeory on the Performance of Optimistic Concurrency Control Schemes," *in Proc. Int. Conf. Databases, Parallel Architectures, and Their Appl. (PARBASE-90)*, Miami Beach, FL, 1990, pp. 86-90.

[4] P. S. Yu, H. Heiss, and D. M. Dias, "Modeling and Analysis of a Timestamp History Based Certification Protocol for Concurrency Control", *IEEE Trans. Know. and Data Eng.*, vol. 3. No. 4, pp. 525-537, Dec. 1991.

[5] H. T. Kung and J. T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Trans. Database Syst.*, vol. 6, No. 2, pp. 213-226, June 1981.

[6] M. J. Carey and M. Livny, "Conflict Detection Tradeoffs for Replicated Data," *ACM Transaction on Database Systems*, vol. 16, pp. 703-746, Dec. 1991.

[7] J. T. Robinson, "Experiments with Trans-

action Processing on Multiprocessor," *IBM Res. Rep. RC9725*, Yorktown heights, NY, Dec. 1982.

[8] P. A. Berstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Reading, MA: Addison-Wesley, 1987.

[9] P. S. Yu and D. M. Dias, "Analysis of Hybrid Concurrency Control Scheme for a High Data Contention Environment," *IEEE Trans. Software Eng.*, 18(2), pp. 118-129, Feb. 1992.

[10] A. Adya, R. Gruber, B. Liskov and U. Maheshwari, "Efficient Optimistic Concurrency Control Using Loosely Synchronized Clock," *In Proc. ACM SIGMOD*, 1995.

[11] M. J. Franklin and M. J. Carey, "Client-Server Caching Revisited," *Proc. of Int. workshop on Distributed Object Management*, 1992.

[12] 김대인, 황부현, "이동 컴퓨팅 환경에서 록 연산과 직렬화 그래프를 이용한 이동 트랜잭션의 직렬성 유지 방법," 정보과학회 논문지, 제26권 9호, pp.1073-1084, 1999.

[13] 최희영, 황부현, "중복 데이터베이스 시스템에서 낙관적인 원자적 방송을 이용한 동시성제어 기법," 정보처리학회 논문지, Vol. 8, No. 5, pp543-552, 2001.

[14] A. Thomasian, "Distributed Optimistic Concurrency Control Methods for High-Performance Transaction Processing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, No. 1, pp.173-189, 1998.

홍 성 찬

1979년~1982년 고려대학교 통계학과(이학사)
1988년~1990년 일본게이오대학 이공학부 관리공학전공(공학석사)
1990년~1994년 일본게이오대학 이공학부 관리공학전공(공학박사)
1993년~1994년 일본 게이오대학 이공학부 객원 연구원
1994년~1995년 LG-EDS시스템(주) 컨설팅 책임연구원
1995년~1996년 상명대학교 정보과학과 전임강사
1997년~현재 한신대학교 정보통신학과 교수
관심분야 : XML, 분산처리, 인터넷비즈니스, 정보시스템 응용