Efficient Optimistic Concurrency Control for Mobile Real-Time Transactions in a Wireless Data Broadcast Environment*

Li Guohui,Yang Bing,Chen Jixiong
School of Computer Science & Technology, Huazhong University of Science & Technology
Wuhan, Hubei Province, 430074
Email: guohuili@hust.edu.cn

Abstract

In this paper, we introduce a variant of the optimistic concurrency control protocols.The broadcast cycle is divided into multiple sub-cycles. Between two sub-cycles, there is a reserved space to accomodate identities for all the data objects which are updated by transactions in the server after the first sub-cycle begins. A read-only mobile transaction can validate its consistency autonomously by comparing its read-set with the committed write-set of the update transactions. If a read-only transaction cannot pass the partial validation, it is not simply aborted and restarted. Instead, an enhanced forward validation policy is applied, which help read-only transactions have more chances to commit. Extensive experiments are conducted to evaluate the performance of the proposed algorithms.

Keywords

Data Broadcast, Optimistic Concurrency Control, Mobile Real-time Transaction Processing, Hybrid Validation Policy

1. Introduction

In a mobile computing system, broadcast-based data dissemination is a major mode of data transfer [1,2]. The transactions processed by mobile clients are called mobile transactions (MT). Some of the mobile transactions can have timing constraints and require the system to finish their processing before their deadlines expire [3] . In this paper, it is assumed that if a mobile real-time transaction (MRTT) misses its deadline, the system just discards the transaction. In this system , many of the MT's are read-only transactions and they do not modify any data items in a database [4]. It is more efficient to process these read-only transaction in a separate way, especially in the wireless data broadcast environment, by taking advantages of the information that they do not update the database.

There are already many studies on data management in a wireless environment [5,6]. Shanmugasundaram proposed a correctness criterion to allow read-only transactions to read current and consistent data without contacting the fixed database server [7]. Victor discussed the disadvantages of Shanmugasundaram's algorithm and showed that two different read-only transactions may perceive the effects of update transactions in different serialization order and this may cause hazard to the system under certain circumstances [8].

In this paper, serializability is adopted as the correctness criterion and efficient control information are broadcast along with the database contents to help mobile clients to process read-only MRTT's without contacting the fixed database server. A variant of the optimistic concurrency control protocol is adopted which is different from the traditional optimistic concurrency control protocols and read-only MRTT's have more chances to commit before their deadlines expire.

2. Optimistic Concurrency Control In Broadcast Environments

In such circumstances where there are less data conflicts among transactions, optimistic concurrency control protocols are adopted to maintain the database consistency.

2.1 Limitations of Existing Optimistic Concurrency Control Protocols for MTs

In the following discussion, the read set of transaction T_i is denoted as read-set(T_i) and the write set of transaction T_j is denoted as write-set(T_j).

* This work is supported by National Science Foundation and a Project Sponsored by the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry

Observation 1. In the backward validation, some conflicts do not affect the correctness of transaction execution.

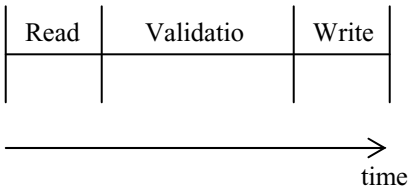

Fig. 1. The three phases of a transaction

As depicted in Fig. 1, when transaction T_j performs a backward validation, since T_i commits during its execution, so T_j validates against T_i and read-set(T_j) is compared with write-set(T_i) and there is an intersection between the two sets. So the validation for T_j fails and T_j aborts and restarts. Similarly, when T_k validates against T_i, because there is an intersection between write-set(T_i) and read-set(T_k), and T_k is aborts and restarts. But actually T_k is different from T_j. The correctness criterion of the optimistic concurrency control protocol requires that the effect of the concurrent execution is equal to that transaction T_j come after T_i. However, T_j reads a data object x before T_i writes x, this conflicts with the correctness criterion. So T_j can't pass the validation and is aborted. Such kind of conflicts which do affect the correctness of concurrent transactions are called **severe conflicts**. On the contrary, even if there is an intersection between write-set(T_i) and read-set(T_k), T_k reads y after T_i writes y, so the **data conflict does not affect the concurrent execution correctness requirements that T_k come after T_i.** And the abortion of transaction T_k is unnecessary because after T_k restarts, T_k will read the same value of y as before (if no other transactions write y). This kind of data conflicts is **fake conflicts**.

Now suppose that T_i is an update transaction executed in the fixed database server, and T_j and T_k are two read-only transactions in mobile clients. T_k reads y after T_i writes y. In other words, the value of y read by T_k is written by T_i. According to Victor's algorithm, before the commitment of T_k, it will find a read-write conflict between T_i and T_k and T_k will be aborted and restarts. However, as in the above discussion, such a conflict is a fake conflict and this transaction abort and restart is unnecessary. So it has a negative effect on system performance.

2.2 Forward Validation Scheme for Mobile Transactions

To differentiate severe conflicts from fake conflicts, when an update transaction in the fixed database server commits, its write set is included in a broadcast cycle immediately. All the transactions in mobile clients compare their read set with the broadcasted write set of the committed update transaction in the fixed DB server.

The set of data objects read by a transaction T_M up to a time point τ is denoted as read-set$_\tau$(T_M). At time τ, an update transaction T_U in the fixed database server commits and its write set is included in the data broadcast cycle. For a mobile transaction T_M and a committed update transaction T_U (Note that the committed write data objects are included in the broadcast cycle), if write-set(T_U)$\cap$read-set$_\tau$(T_M) $\neq\Phi$, then T_M aborts and restarts. Such a comparison is called a partial validation for a mobile transaction.

For a read-only mobile transaction T_M, if it passes all its partial validations during its execution, it is ready to commit. Before the final commit, it waits for the commit of the next update transaction $T_{U'}$, if write-set(T_U)$\cap$read-set(T_M) =Φ, then T_M is committed and the query results are returned to the user. Otherwise, T_M aborts and restarts.

For an update transaction T_{MU}, it is transmitted to the fixed database server for final validation. Even it is necessary to transfer mobile update transactions to the fixed DB server via the upstream bandwidth. By using the partial validation scheme, a destined-to-fail transaction aborts and restarts in advance and it is unnecessary to transfer them for final validation. This can increase the transaction success ratio.

2.3 Enhanced Validation Policy

The validation policy proposed in section 2.2 cannot predict when an update transaction in the fixed DB server commits and the number of data items an update transaction can modify. So a mobile client has to stay in active state and tune in to get the interesting data items continuously. This increases the power consumption for mobile clients tremendously.

To solve this problem, we design a new data broadcast method. A broadcast period is divided into several broadcast sub-periods (SP) which are of the same length. Between two SP's, we reserve a space with a fixed size for the set of data items which are updated by transactions in the fixed database server. Such a space is reserved mainly for mobile transactions to validate themselves autonomously. In the middle of two consecutive broadcast SP's, the set of identities for all the data items that are updated in the fixed database server after the first broadcast SP is included in the broadcast channel.

The size of the reserved space between two consecutive broadcast SP's has a subtle impact on the system performance. If the size of the reserved space is too large, of course, more transaction aborts caused by fake conflicts can be avoided. However, large size of the reserved space means an extended data broadcast

period. This can increase the data accessing time latency for mobile transactions.

Since the size of the reserved space between two consecutive broadcast SP's are predefined, it is possible that the number of the data items updated by transactions in the fixed database server between two SP's exceeds the reserved space between the two broadcast SP's, i.e., the only penalty for such reserved space inadequacy is that some fake conflicts cannot be differentiated and thus some transactions are aborted unnecessarily due to the fake conflicts.

Suppose that the set of committed transactions which commit after the previous broadcast sub-cycle is denoted as T. $T=\{t_i|1\le i\le n\}$ and transactions in T are sorted according to the committing order, i.e., transaction t_i commits before t_{i+1} commits $(1\le i\le n-1)$. Suppose that:

$$\sum_{i=1}^{j} space(ID(write-set(t_i))) \le reserved-space$$

and

$$\sum_{i=1}^{j+1} space(ID(write-set(t_i))) > reserved-space$$

The set of transactions in T are partitioned into two sub-sets: $T1=\{t_1, t_2, \ldots, t_j\}$ and $T2=\{t_{j+1}, t_{j+2}, \ldots, t_n\}$, i.e., $T=T1\cup T2$. Note that T2 can be an empty transaction set. After the partitioning, the identities of all the data items which are updated by any transaction in T1 are included in the reserved space after the current broadcast sub-cycle. The identities of all the data items which are updated by any transaction in T2 are included in the next reserved space. Furthermore, a flag (bit) is included in the reserved space to specify whether there is enough space to accommodate all the identities of the data items updated by all the transactions in the fixed database server that commit in the previous data broadcast sub-cycle.

The main steps executed by a mobile transaction are described as follows:

(1)Based on the data index preceding a data broadcast cycle, a mobile transaction determines the relative position of the interesting data items in the broadcast cycle. Then the mobile client turns into the doze state.

(2)Before an interesting data item appears in the broadcast channel, the mobile client turns from the doze state to the active state and then accesses the data items.

(3)When there comes the reserved space between two consecutive data broadcast sub-cycle, the mobile client turns from the doze state to the active state and then extracts the set of identities for the updated data items and the flag in the reserved space.

(4)The mobile client compares the set of data items included in the current reserved space, denoted as WS, with the set of data items read by the mobile transaction, TM, up to now-the time point is τ

if $WS\cap$read-set$_\tau$(TM) $\neq\Phi$

THEN Transaction TM aborts and restarts

ELSE TM continues its execution.

Compared with the algorithm in section 2.2, our algorithm is more power-efficient. By using the data index, a mobile client can turns into the doze state before the interesting data appear in the wireless broadcast channel. However, in section 2.2, we cannot predict when a transaction in the fixed database server commits and a mobile client has to stay in active state and tunes in to access the interesting data.

3. Performance Evaluation

The simulation experiments are aimed at studying the performance of our proposed protocol in contrast with the conventional OCC protocol and protocol in [8] in real-time broadcast disk environments. The major performance metric of these protocols is the miss rate, which is the percentage of transactions missing their deadlines. Another performance metric is the restart rate, which is the average number of aborts and restarts before a transaction can be committed. The statistics of read-only mobile transactions (ROMT) and update mobile transactions (UMT) under conventional OCC protocol in [8] (FBOCC) and our proposed protocol (EOCC) are collected separately.

Fig. 2 shows the restart rate of mobile transactions under different concurrency control protocols.

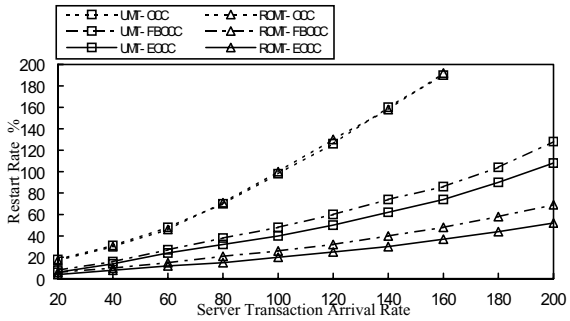

Fig.2. Restart rate versus server transaction arrival rate

Note that a transaction may be restarted more than once. OCC owns the highest restart rate and response time for all of the mobile transactions must be sent to fixed host to validate. FBOCC reduces both performance metrics by utilizing partial validation. For our proposed protocol, the performance is similar to that of FBOCC in case of low data contention. However, with the increase of server transaction arrival rate, our protocol does better than FBOCC because the enhanced validation method can differentiate between severe conflicts and fake

conflicts Fig. 3 illustrates the miss rates of several protocols.

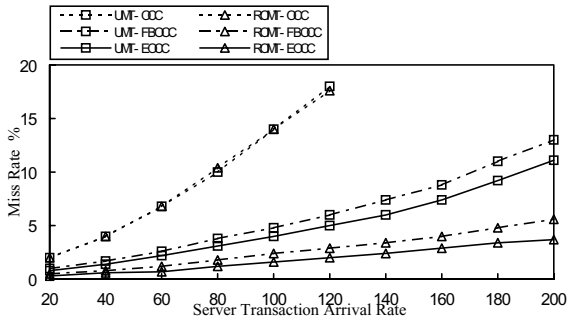

Fig.3. Miss rate versus server transaction arrival rate

For OCC, there is no discrimination between ROMT and UMT. Both are required to submit to the server for validation. Therefore, the performance of these two transaction types is almost the same. The miss rate increases with the server transaction arrival rate. Although the load at the server is not high such that the impact of resource contention is not great, the increasing number of server transactions increases the chance of data conflicts between mobile transactions and server transactions. Consequently, mobile transactions submitted to the server for validation will be restarted if any data objects that have been read are over-written by a committed sever transaction. Since the delay overhead in broadcast environments is much higher than that in wired systems, it is more likely for a restarted transaction to miss its deadline. The performance of FBOCC is better than traditional OCC thanks to the autonomous partial validation at mobile clients. The performance of ROMT is even better than that of UMT since the saving of validation of ROMT at the server helps them to meet more deadlines. For our proposed protocol, the performances of ROMT and UMT both outperform those of FBOCC and traditional OCC. The main reasons are that fake conflicts can be differentiated from serve conflicts by the enhanced validation method. Moreover, the dynamic serialization order adjustment of read-only transactions further reduces the probability of missing deadlines for ROMT.

4. Conclusion

The proposed algorithm in this paper supports autonomous read-only mobile transaction processing without contacting the fixed database server. Furthermore, we can make difference between severe conflicts and fake conflicts. This results in more possibility of successful commitment for mobile transactions.

Furthermore, for a mobile read-only transaction, we combine the backward validation and forward validation to achieve more transaction commitment a read-only transaction has more its chance to commit correctly. In the broadcast cycle, we can preserve spaces to accommodate the write sets of the committed update transaction in the fixed database server.

In this paper, a data broadcast cycle is divided into a pre-defined number of sub-cycles and the size for the space reserved between two consecutive sub-cycles is also fixed. Actually, the number of sub-cycles and the size for the reserved space between two sub-cycles have an impact on the system performance. We are now focusing on this problem in both analytical and experimental ways.

6. References

[1] R. Alonso, H. Korth, Database systems issues in nomadic computing, Proc. Of the ACM SIGMOD Conference, Washington D. C., June 1993, pp. 388-392

[2] T. Imielinski and B. r. Badrinath, Mobile wireless computing: challenges in data management, Communications of the ACM, Vol. 37, No. 10, October 1994, pp18-28

[3] Stabjivic, J. A., Son, S. H., Hansson, J., Misconceptions about real-time databases, Computer 199, 32(6), pp29-37

[4] Garcia-Molina, H., Wiederhold, G., Read-only transactions in a distributed database, ACM Transactions on Database Systems, 1982,7(2): 209-234

[5] Barbara, D., Certification reports: supporting transactions in wireless systems, In: Proceedings of 17th International Conference on Distributed Computing System, USA, 1997: 466-473

[6] Acharya, S., Alonso, R., Franklin, M., Zdonik, S., Broadcast disks: data management for asymmetric communication environments, In: Proceedings of the ACM SIGMOD conference, 1995: 199-210

[7] Shanmugasundaram J., Nithrakashyap, A., Sivasankaran, R., Ramamritham, K., Efficient concurrency control for broadcast environments. In: ACM SIGMOD International Conference on Management of Data. 1999

[8] Victor C. S. Lee, Kwok Wa Lam, Tei-Wei Kuo, Efficient validation of mobile transactions in wireless environments, Journal of Systems and Software, 2004.1