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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.
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 $WT_i$ 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 $WT$s, 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 $WT_i(RD_i), WT_2(RD_i), ..., WT_k(RD_i)$. If it can’t find some $WT_i(RD_i)$ equal to the version read for any $RD_i$, the transaction is aborted. Otherwise, the valid interval of $RD_i$ is $(WT_i(RD_i), WT_{i-1}(RD))$ where $WT_i(RD_i)$ is equal to the version number accessed for $RD_i$. The valid interval of write timestamp $WT_i(RD_i)$ is $(WT_i(RD_i), ST)$. 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 \( Ti \) in set \( CS \). Therefore, the set \( CS \) consists of \( (Ti, MT_i) \) 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 \( D_i \). The set \( RS \) is maintained to update the timestamp \( MT \) and the set \( WS \) is maintained to decide whether to abort. When transaction \( Ti \) is started, the server inserts the transaction identifier \( Ti \) into set \( CS \) and sets timestamp \( MT_i \) of the transaction \( Ti \) to timestamp \( RT_i \). Whenever transaction \( Ti \) reads each data item \( D_k \), transaction identifier \( Ti \) is inserted into set \( RS_i \). Whenever transaction \( Ti \) 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 \( Ti \) has accessed any write-write conflicting data item or not. If it has accessed conflicting data items, then transaction \( Ti \) is aborted. However, if it has not accessed conflicting data items, then transaction identifier \( Ti \) is inserted into set \( WS_i \).

When transaction \( Ti \) requests a commit, we use the validation algorithm in[4]. After transaction \( Ti \) passed the validation phase, we try to find conflicting transactions based on set \( RS \) and \( WS \). For each data \( D_k \) that was read and written by transaction \( Ti \), if a transaction identifier is in set \( RS \) and \( WS \), the transaction is aborted. After all conflicting transactions have been aborted, all timestamp \( MT_i \) is updated only if transaction identifier \( Ti \) is in set \( RSi \) of data \( D_k \) that was written by transaction \( Ti \). Transaction \( Ti \) is committed after all information of transaction \( Ti \) have been deleted from set \( CS \), \( RSi \) and \( WSi \).

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 piggybacked 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 piggybacked 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 \( Ti \) 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 $T_i$. For each accessed data $D_k$ by committing transaction $T_i$, if transaction identifier $T_i$ 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 $ST_j$, each timestamp $MT_i$ is set to timestamp $ST_j$ only if timestamp $MT_i = RT_i$, where $MT_i$ denotes the maximum timestamp of the transaction identifier $T_i$ that is in set $RS_k$ 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 $MT_i$ will be set to a re-ordered timestamp $OT$ only if timestamp $MT_i = RT_i$ or $MT_i > OT$, where $MT_i$ denotes the maximum timestamp of transaction identifier $T_i$ that is in set $RS_k$ associated with data $WD_k$ updated by re-ordered transaction $T_i$. Therefore, the value of maximum timestamp $MT$ is never increased after the value of $MT$ is changed from $RT_i$. After this happens, all transaction identifiers $T_i$ 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 to 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
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 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 affect the number of re-ordered transactions.

Fig. 3 shows 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

Table 1. Parameters, Settings and their meanings

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Settings</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num_Data</td>
<td>1000</td>
<td>Number of data</td>
</tr>
<tr>
<td>Num_Trans</td>
<td>10 ~ 25</td>
<td>Number of transactions</td>
</tr>
<tr>
<td>OP</td>
<td>10, 20, 30</td>
<td>Number of operations</td>
</tr>
<tr>
<td>Per_Write</td>
<td>20%</td>
<td>Percentage of write probability</td>
</tr>
<tr>
<td>Time_Read</td>
<td>20 time units</td>
<td>Read access time</td>
</tr>
<tr>
<td>Time_Write</td>
<td>30 time units</td>
<td>Write access time</td>
</tr>
<tr>
<td>Time_Net</td>
<td>40 time units</td>
<td>Network delay</td>
</tr>
<tr>
<td>Time_Restart</td>
<td>20 time units</td>
<td>Re-start delay</td>
</tr>
<tr>
<td>Time_Valid</td>
<td>Num_Oper X Time_Read</td>
<td>Validation time</td>
</tr>
<tr>
<td>Hist_Size</td>
<td>3</td>
<td>Size of Write History</td>
</tr>
</tbody>
</table>

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


