Optimistic Scheduling for Transaction Management in Mobile Database Systems

MinKyo LEE¹, Nonmember, JongHyun LEE², and Songchun MOON¹, Members

SUMMARY  In mobile computing environment, in which communication channels are limited and have low-bandwidths, mobile transactions have long-lived and frequently disconnected with their wireless network in processing. Such peculiarities of mobile transactions make existing transaction scheduling schemes inadequate and raise new challenging research problems. In this paper, we propose a new scheduling scheme called OTS/MT (Optimistic Timestamp Scheme for Mobile Transactions) for mobile transaction scheduling. OTS/MT is based on optimistic approach that is suitable for low data contention, and prevents indefinite postponement and cascading delay which are major drawbacks of existing optimistic concurrency control scheme and timestamp ordering scheme. In addition, OTS/MT algorithm is inherently a deadlock-free scheduling scheme. In order to schedule mobile transactions, OTS/MT postpones the detection of conflict between mobile transactions until transaction commit time to improve the performance deterioration of TO. In this paper, we attempt to show that this application of optimism to TO is justified by a way of simulation.

key words: Mobile Database, mobile transaction, optimistic scheduling, Concurrency control

1. Introduction

Recent advances in network technologies have made it possible that users can access information through wireless connection and move into another location while preserving its wireless link. Generally, such a mobile computing environment consists of two distinct sets of entities: mobile hosts and fixed hosts. Mobile hosts(MHs) can unrestrictedly move either within a cell which implies a radio coverage area or between two cells while retaining their network connection. In addition, MHs can connect to their wireless network from different locations at any times. The hosts other than mobile ones are steadily connected with a stationary network and some of them, called mobile support stations(MSSs), are augmented with a wireless interface to communicate with MHs.

Using MHs, users are allowed to request information to MSSs and receive realtimed responses during their movement. In order to access information through wireless connections, users normally issue queries, which we will denote as mobile transactions.

¹The authors are with the Department of Management Engineering Graduate School of Management, KAIST, Cheongryang, Dongdaemoon, Seoul 130-012 Korea.

²The authors is with Logistics Technology Team of Multimedia Technology Lab., Korea Telecom, 17 Woomyun Sucho, Seoul 137-792 Korea.

A mobile transaction is a sort of distributed transaction in its nature, where some parts of the computation are executed on MHs and others on fixed hosts. The use of the wireless medium and the resulting mobility of data consumers and producers introduce new technical issues in transaction processing part of database systems.

Due to potentially long network delays, the employment of wireless connections results in transactions which are expected to be long lived, if they are executed in distributed systems based on fixed networks, although the transactions are not exactly the case. Moreover, during transaction execution mobile hosts are vulnerable to be disconnected with their wireless network voluntarily or involuntarily. The reason is that users of mobile hosts often avoid the employment of the wireless medium because wireless connections are expensive both monetarily and in terms of power consumption, or because no networking capability is available at their current location. When being disconnected, MHs may execute some or whole parts of transactions themselves with data cached in their local disks. After completing the transactions, MHs may send updated data objects, if any, to MSSs in order to reflect the execution results of the transactions to corresponding databases.

Such peculiarities of mobile transactions in mobile computing environment make optimistic approach for transaction scheduling more suitable than pessimistic approach. For instance, although two-phase locking method is a typical scheme that is widely used for transaction scheduling in distributed database systems, from the performance point of view, it is intrinsically inappropriate for long-lived transactions because of exclusive and pessimistic properties associated with locking mechanism. In addition, in traditional distributed database systems, disconnections happening during transaction execution have been treated as failures or crashes. In mobile computing environment, however, these approaches are no longer practical. If disconnections happening in transaction execution are treated as failures, the recovery cost of transactions may excessively increase due to their frequency. As a consequence, pessimistic scheduling methods designed for traditional distributed transactions may be less eligible to mobile computing environment than optimistic methods. Therefore, many scheduling schemes choose
An optimistic approach to handle mobile transactions effectively in mobile computing environment.

In this paper, we attempted to prove that optimistic way of control and deadlock-freeness are the two major properties to make the management of mobile transaction realistic. The existing optimistic concurrency control (OCC) scheme could incur a serious performance degradation when MHs are disconnected. Because of large quantity of communication for maintaining lock and difficulty for detecting global deadlock, the locking mechanism has a weakness in mobile environment. Timestamp ordering scheduling scheme, TO for short, which is a classical deadlock-free one has a considerable performance deterioration by its pessimistic property. Therefore, we propose a new scheduling scheme called Optimistic Timestamp Scheme for Mobile Transactions (OTS/MT) for short, for mobile transaction scheduling. OTS/MT basically takes an optimistic way of control in the sense that it postpones the detection of conflict between mobile transactions until transaction commit time to improve the performance deterioration of TO.

The remainder of this paper is organized as follows. Section 2 describes problems of cascading delay and indefinite postponement in previous works. In Section 3, we describe OTS/MT algorithm in detail and the correctness of the proposed algorithm. Section 4 presents an analytic model of OTS/MT for performance evaluation. Section 5 describes the performance comparison between our approach, TO and OCC. Finally, conclusions appear in Section 6.

2. Related Work and Problem Definition

As an optimistic scheduling scheme that is applicable to mobile distributed database system (MDDSs), there is O2PL (Optimistic Two Phase Locking) method[6]. Originally, O2PL is designed for an optimistic read-only write-all concurrency control in replicated database systems in which multiple copies of some data objects are stored at multiple sites. In O2PL, a read lock must be obtained immediately from the local or nearest copy site for read operation. Write locks for replicated copies are deferred until the beginning of the commit phase is reached. The basic idea underlying O2PL is to set locks immediately within a site if data objects are replicated in the site, while taking a more optimistic and less message-intensive approach across site boundaries. In mobile computing environment, however, O2PL may increase message costs for the lock management due to the mobility of transaction hosts, as indicated in [8]. In order to reduce the message costs incurred by the mobility of transaction hosts, Jing[8] has proposed O2PL/MT (Optimistic Two Phase Locking for Mobile Transactions) which is a variant of O2PL method. Rather than sending a read unlock to the remote copy server where the read lock is set, O2PL/MT simply allows the unlock operation to be executed at the local or nearest copy server of data object.

Although O2PL/MT contributes towards reducing the read unlock message cost incurred by the mobility of transaction hosts, there are some inherent drawbacks. First, in case that locks on data objects are long-pending by a mobile transaction in disconnection state, the data objects may be subject to lengthy blocks and, consequently, the execution of other transactions which would like to access the data objects will be suspended until the mobile transaction releases the locks on the data objects. Second, since O2PL/MT does not consider the situation of disconnection, the single disconnection MH that has locked data items may cause continuous transactions' aborts. If the disconnection exists, the detection and solving deadlock will become more complicated. Moreover, by using lock mechanism, O2PL/MT could not have the chance of decreasing the number of message exchange for the validating a transaction. Therefore, deadlock-free concurrency control methods are preferred to other methods in mobile computing environment. As deadlock-free concurrency control methods, there are two typical schemes: optimistic concurrency control scheme and timestamp ordering scheme[4].

Optimistic concurrency control (OCC) scheme requires the execution of transactions to be divided in three phases: the read phase, the validation phase, and the write phase. In the read phase, a transaction does all its work. Next, in the validation phase, a conflict test is performed to determine if the transaction violates consistency. If there are no possible inconsistencies, the transaction enters its write phase and it is committed. If the transaction fails validation, it is aborted and restarted. Although OCC scheme is basically an optimistic scheduling scheme, it could take a serious performance degradation as shown in Example 1 in case that OCC scheme is employed as a scheduling method for transactions in a mobile environment.

Example 1 (Indefinite Postponement in Validation Phase of OCC Scheme):

Assume that an MSS happens to validate a mobile transaction MT1 at time t1 with other three mobile transactions, MT1, MT2, and MT3, each of which completed its local execution before time t1 (Figure 1). Let Wm1(x) and R1(x) be a write operation and a read operation of a transaction MT1, respectively, on a cached data object z and let WC1(x) be a write operation sent by a MH to the MSS for committing MT1. To validate MT1, the MSS should know the write-data sets of MT1, MT2, and MT3. In case that the commitment of MT2 is delayed due to some reasons, e.g. unpredictable disconnections or low-bandwidth channels, the MSS should wait until time t2 when transferring the write-data set of MT2 is completed. That is, validating MT4 should be delayed at least after time t2.
End of Example □

In timestamp ordering (TO) scheme, a unique timestamp is assigned to each transaction by a transaction manager when it is submitted to the transaction manager. According to the assigned timestamps, a transaction scheduler orders conflicting operations issued by transactions. In TO scheme, the detection of conflicts is basically accomplished in the phase of receiving operations from transactions. However, in a mobile environment, if a timestamp is assigned to a mobile transaction and shortly the MH of the transaction falls into lengthy disconnection state, the execution of other transactions which conflict with the mobile transaction should be also delayed until the mobile transaction is either committed or aborted. This may result in considerable performance deterioration of MDDS. Example 2 illustrates such a problem.

Example 2 (Cascading Delay Due to Disconnection in TO Scheme):
Assume that an MSS happens to schedule three mobile transactions, MT₁, MT₂, and MT₃. Let the three transactions be defined as follows:
MT₁: R₁(x)W₁(x)R₁(y)W₁(y)commit₁,
MT₂: R₂(s)W₂(u)R₂(x)W₂(z)commit₂,
MT₃: R₃(s)W₃(u)R₃(x)W₃(z)commit₃
In addition, suppose that the MH of MT₁ is disconnected from the MSS at time t₁ and reconnected at time t₂ (Figure 2). For simplicity, assume that the timestamp of each transaction is equal to its subscript (i.e. timestamp (MT₁) = 1). At time t₂, R₂(x) is success-

sent them. The scheduler of TO puts R₂(x) into waiting queue of x instead of allowing to execute. In waiting queue of x, R₂(x) has to be waited until write-transit[x] is set to be 0. Since read-scheduled[x] is 1, W₂(x) is scheduled by the scheduler. Seeing that R₂(x) is waiting in the queue, W₂(x) is added to the next of R₂(x) in the waiting queue of x. write-transit[x] is 1 through the disconnection time of MT₁ and is set to be 0 by MT₁ when it is reconnected. Finally, MT₁ is reconnected at time t₄ and R₂(x) is executed. Therefore, R₃(x) has been waited until W₁(x) is completed. In case of MT₃, R₃(x) is delayed to time t₅ by the delay of W₂(x).

In this case, both the transactions MT₂ and MT₃ should be successively long delayed till the preceding transaction, MT₁ is completed (i.e. time t₄) since ts(MT₁) < ts(MT₂) < ts(MT₃) and they are in conflict.

End of Example □

In Example 2, actually the MSS can not be aware of the reconnecting time of MT₁. Accordingly, under such a condition as the reconnecting time is unknown, it is not desirable that other transactions in conflict haphazardly wait for MT₁ to finish.

The problems of indefinite delays and cascading delays in the previous validation schemes are mainly caused from that it does not consider the mobile characteristics. In the mobile environment, the execution order of transactions is liable to be different from their commit order due to the data transfer delay through the wireless channel. Thus, the validation scheme which considers only the local execution order is susceptible to the problems of indefinite delays and cascading delays resulting from the gap between the two different order. In this respect, a new validation scheme needs to be devised to minimize the effect caused by the gap between the two orders and it is needed to be optimistic approach.

3. Optimistic Timestamp Scheme for Mobile Transactions: OTS/MR

3.1 Mobile Database System Model

In this section, we present a mobile database system model for our discussion. Figure 3 illustrates a general mobile database system model for mobile computing environment. The overall network architecture consists of a wired network of fixed hosts and low-bandwidth wireless cells, each comprising of a fixed host called MSS and MHs. A mobile host can directly communicate with an MSS within its cell via a wireless medium. In this model, both a database server and a database are attached to each MSS. A database server is to support basic transaction operations such as read, write, prepare, commit, and abort.

A MH submits mobile transaction operations to
MSS which takes charge of current cell. Then, the MSS in turn sends them to the distributed database server within the fixed network for execution. In addition, each MSS monitors their execution in database servers within the fixed network. A mobile transaction consists of a set of read and write operations started by a begin statement and terminated by a commit or an abort statement. Mobile transactions may be submitted in one of two fashion. As the first one, an entire mobile transaction may be submitted in a single request message. Thus, the whole mobile transaction becomes one submission unit. The MH also delivers execution control to its MSS and awaits the return of the results of the transaction execution. On the other hand, the operations of a mobile transaction may be submitted in multiple request messages. Accordingly, a submission unit consists of one operation (e.g. read) or a group of operations. The MH interactively submits the operations of a mobile transaction to its MSS. A subsequent operation can be submitted only after its previous one has been executed and the result returned from the MSS. In this case, multiple MSSs may be involved for the execution of the mobile transaction because of the mobility of the host. For example, a MH may move into a new cell after it obtains the results of previously submitted operations. In the new cell, it will submit the remainder of the transaction operations to the new MSS. Our proposed model employs the second approach to transaction submissions. This approach supports the interactive execution of transactions and therefore offers increased flexibility in transaction computations.

3.2 Optimistic Timestamp Scheduling

In OTS/MT, a transaction manager on MSS assigns a unique timestamp, ts(MT_i), to each mobile transaction, MT_i. Timestamps generated by the transaction manager are in monotonously increasing order. In order to resolve data conflicts, OTS/MT maintains for every data object x the maximum timestamps of Read and Write on x, denoted max-read-ts[x] and max-write-ts[x]. The max-read-ts[x] and max-write-ts[x] designate to the most recent timestamps for read and write that accessed the data object x. In this way, the influence of a transaction that has successfully committed is imprinted to all the data objects it accessed. A transaction scheduler in OTS/MT schedules operations issued from mobile transactions without delay. That is, when the scheduler receives an operation from the transaction manager, it does not decide whether to accept or reject it, but immediately schedule it. Note that, in traditional TO scheme, whenever the scheduler receives an operation, the scheduler should determine whether to accept, or reject it.

When a mobile transaction, MT_i, is ready to enter its commit phase, the scheduler checks whether the values of max-read-ts and max-write-ts on data objects which are read and written by MT_i are updated by other transactions since the data objects are accessed by MT_i. If so, the scheduler rejects the commit operation of MT_i and aborts MT_i because data objects accessed by MT_i are updated by some transactions. Otherwise it certifies MT_i by passing the commit operation to a data manager, thereby allowing MT_i to terminate successfully.

In order to check whether an MT can be safely committed or must be rejected, OTS/MT employs several data structures: a set containing the names and timestamps of active transactions, and two sets, read-scheduled[MT_i] and write-scheduled[MT_i], for each active transaction MT_i, which include not only the data objects read and written by MT_i, but also the values of max-read-ts and max-write-ts of the corresponding data objects. When the scheduler receives a read operation, r(x) or write operation, w(x) of MT_i, it adds x and the value of max-read-ts[x] (or max-write-ts[x]) to read-scheduled[MT_i] (or write-scheduled[MT_i]). When the scheduler receives commit_i, i.e. MT_i has finished executing, so read-scheduled[MT_i] and write-scheduled[MT_i] become to contain MT_i’s read set and write set, respectively. Thus, testing for conflicts can be done by comparing the value of max-read-ts or max-write-ts for each data object x in read-scheduled[MT_i] or write-scheduled[MT_i] with current value of its max-read-ts or max-write-ts. As described earlier, the current value of max-read-ts[x] and max-write-ts[x] for every data object x mean the most recent timestamp when the data object x is read and written by transactions.

To resolve read-write conflict, the value of max-write-ts for each data object x in read-scheduled[MT_i] is compared with current value of its max-write-ts. If the value of max-write-ts[x] in read-scheduled[MT_i] is identical with current value of its max-write-ts, the Read operation on the data object can be considered to have been correctly executed. Otherwise, MT_i should be aborted because the operation has been in-
correctly executed. In order to check both write-read and write-write conflicts, the stored values of max-reads and max-write-ts for each data object \( x \) in write-scheduled[\( MT_i \)] are compared with current values of both max-reads and max-write-ts of \( x \), respectively. If the values of max-read-ts[\( x \)] and max-write-ts[\( x \)] in write-scheduled[\( MT_i \)] are equivalent to current values of its max-reads and max-write-ts, the Write operation on the data object can be taken as having been correctly executed. Otherwise, \( MT_i \) is aborted by the scheduler.

The following algorithm (figure 4) shows the optimistic \( \text{Algorithm (OTS/MT)} \):

**Algorithm (OTS/MT):**

**Input:**
- from Transaction Manager: transaction.data object to access, its timestamp;
- from Data Manager: "timestamp is assigned by transaction manager";

**Output:**
- to Transaction Manager: validation result;
- to Data Manager: operation on transactions, data object to access;

**Begin**

- label: main;
- receive (operation.type, x, data object to access);
- switch(operation.type)
  - case begin:
    - receive (timestamp, x);
    - "timestamp is assigned for the whole transaction";
    - add transaction_id to active transaction set;
  - case read:
    - read_scheduled() = read_scheduled() + x;
    - add data object to the read_scheduled set of the transaction /send op(x) to data manager and receive max_write_ts(x) from data manager/;
    - the_max_max_write_ts(x) = max(max_write_ts(x), the_max_max_write_ts(x));
  - case write:
    - write_scheduled() = write_scheduled() + x;
    - add data object to the write_scheduled set of the transaction /send op(x) to data manager and receive max_write_ts(x) from data manager/;
    - save the values of max_read_ts(x) of the data object /the_max_max_write_ts(x) = max(max_write_ts(x), the_max_max_write_ts(x))/;
  - case commit:
    - valid = true;
    - for \( x \) in read_scheduled() do
      - if the_max_max_write_ts(x) >= max_write_ts(x) then
        - valid = false /validation failed/;
    - end_for;
    - for \( x \) in write_scheduled() do
      - if [the_max_max_write_ts(x) >= max_write_ts(x)] then
        - valid = false /validation failed/;
    - end_for;
    - return_transaction_manager(revalid);
    - return the result of validation /revalid/;
  - case abort:
    - the_max_max_write_ts(x) = max(max_write_ts(x), the_max_max_write_ts(x));
- end_switch;
- go to start;
- End

**Fig. 4** Algorithm of OTS/MT

timestamp concurrency control algorithm, which are executed by the scheduler on the server of each MSS. In the algorithm, the grayed part indicates a critical section.

In OTS/MT, the problem like the indefinite postponement in OCC cannot inherently be happened because OTS/MT is able to perform the validation test of a committing transaction without being aware of the complete sets on read-data and write-data of currently active transactions. Example 3 shows how the cascading delay, as described Example 2, of mobile transactions can be obviated by OTS/MT.

**Example 3** (Preclusion of Cascading Delay by OTS/MT):

In Example 2, when operations of the transactions \( MT_2 \) and \( MT_3 \) are wholly executed, OTS/MT accomplishes validation tests for their commitment with their saved values of max-read-ts and max-write-ts (the then_max_read_ts and the then_max_write_ts in the OTS/MT Algorithm) on accessed data object, i.e., \( x \) even though the MH of \( MT_1 \) is disconnected from the MSS at time \( t_1 \). Since the saved values of max-read-ts and max-write-ts on \( x \) is equivalent with the current value of max-read-ts and max-write-ts on the data object, both \( MT_2 \) and \( MT_3 \) can be committed by OTS/MT without waiting for \( MT_1 \) to reconnect and be finished. After \( MT_2 \) and \( MT_3 \) are committed, the values of both max-reads and max-write-ts on \( x \) is updated with the timestamps of \( MT_2 \) and finally updated to the timestamp of \( MT_3 \). However, when \( MT_1 \) enters in the commit phase, \( MT_1 \) will be aborted by OTS/MT because the current values of max-read-ts and max-write-ts on \( x \) has been updated by \( MT_2 \) and \( MT_3 \).

End of Example 0

As a result, committed transactions lined up as timestamp order in OTS/MT. OTS/MT could be thought as a version of TO Certifier[4] because of its optimistic approach and validating with timestamp order. When basic TO finds a violation of the TO rule, it immediately rejects the operation, whereas the OTS/MT delays this rejection until it rejects the transaction's commits. Since, allowing such a transaction to complete involves extra work, basic TO is preferable to a OTS/MT in conventional database system. However, basic TO doesn't consider the situation of disconnection, it has a drawback by disconnected transaction which may not violate in fixed network environment. In Example 2, any transaction didn't violate timestamp order but a disconnected transaction blocks other transactions. In Example 3, OTS/MT sacrifices the disconnected transaction to abort. Therefore, OTS/MT can get better throughput even though it aborts more transactions in mobile environment.

### 3.3 Correctness of OTS/MT

The correctness criteria for concurrency control algorithms is based on their serializability[4], which ensures the serializable execution of transactions. Since every serializable execution has the same effect as some serial execution, serializable executions can avoid both lost updates and inconsistent retrievals. In order to prove that the OTS/MT algorithm only produces serializable executions, we will follow the usual procedure of prov-
ing that every history it allows must have an acyclic serialization graph (SG).

The following theorem formally proves that the OTS/MT algorithm produces serializable executions of transactions.

**Theorem (Correctness of OTS/MT):** OTS/MT produces serializable histories.

**Proof:** Let \( H \) be a history representing an execution produced by OTS/MT. Assume that \( T_j \rightarrow T_i \) (where \( i \neq j \)) is an edge of Serialization Graph (\( H \)). Then \( T_i \) and \( T_j \) are committed in \( H \) and there are conflicting operations \( q_j[z] \) and \( p_i[z] \) such that \( q_j[z] \) is executed before \( p_i[z] \) in \( H \). In addition, the value of \( \text{max} \cdot \text{op} \cdot \text{ts}[z] \), where \( \text{op} \) stands for read or write operation, is set with the timestamp of \( T_i \) because \( p_i[z] \) is the latest operation on \( z \). We claim that the certification of \( T_j \) preceded the certification of \( T_i \).

Suppose that \( T_i \) was certified before \( T_j \). At the time \( T_i \) is certified, \( p_i[z] \) must have been processed by the scheduler. That is, \( z \) is in \( \text{op}\cdot\text{scheduled}[i] \) and the value of \( \text{max} \cdot \text{op} \cdot \text{ts}[z] \) must have been also saved (in our algorithm, the \( \text{the}\_\text{then}\_\text{max} \cdot \text{op} \cdot \text{ts}[z] \) is used). At the certification time, the scheduler will compare the values of the saved \( \text{max} \cdot \text{op} \cdot \text{ts}[z] \) and the current \( \text{max} \cdot \text{op} \cdot \text{ts}[z] \) and commit \( T_i \) since the values are equivalent, i.e., if \( \text{max} \cdot \text{op} \cdot \text{ts}[z] \) is not updated by other transactions. However, when \( T_j \) is in certification phase, the scheduler will reject \( T_j \) because the saved value of \( \text{max} \cdot \text{op} \cdot \text{ts}[z] \) of \( T_j \) is not identical with the value of current \( \text{max} \cdot \text{op} \cdot \text{ts}[z] \). This is contrary to our assumption that \( T_j \) is committed in \( H \). By induction, if a cycle existed in the Serialization Graph (\( H \)), every transaction on the cycle would have to be certified before itself. This is an absurdity. Therefore, the Serialization Graph (\( H \)) is acyclic and \( H \) is a serializable history.

End of Theorem \( \Box \)

### 3.4 Application of OTS/MT

Several of the applications envisioned for mobile computing will appear similar to those currently served by laptop computers. These applications could be classified to three categories: **travelers support applications** which will make possible personal communication including information browsing and data entry to database, **distributed work support applications** which will be used in Computer Supported Cooperative Work (CSCW) System or multiperson interaction, and real **time system support applications** which will be used for automatic supervision of airspace or automatic observation of railroad traffic[2][10][14].

These applications could be implemented by concept of mobile database and mobile transaction processing. From a standpoint of transaction processing, two characteristics have to be considered: **mobility** and **connectivity**. Since information could be lost when wireless communication channel is disconnected, managing disconnect situation is more important for distributed work support applications and real time support applications. In case of travelers support applications, mobility is considered more essential characteristic than connectivity. While 02PL/MT deals mobility of transaction more importantly than connectivity, OTS/MT emphasizes improving disconnect situation. Therefore, OTS/MT is more suitable for distributed work support applications and real time system support applications and 02PL/MT is adequate to travelers support applications.

### 4. Analytical Model of OTS/MT

In this section, we discuss an analytical model of OTS/MT. As discussed in Section 3.1, there are two patterns in the execution model of mobile transactions. For the sake of simplicity, we only consider the case that operations of a mobile transaction are submitted in multiple request message. Moreover, we take no thought of the dynamic factors (e.g., channel bandwidth or availability, consistency of cache or replicated data) of mobile networks, and solely focus the mean response time of mobile transactions.

We first describe the mobile transaction model. The mobile transaction model consists of \( N_{L} + 2 \) states, where \( N_{L} \) is the number of data objects accessed, i.e., the mobile transaction size. The mobile transaction has an initial setup phase. This is denoted as state 0, and modelled as \( T_{\text{INIT}} \) to the transaction response time. Following the initial setup, a mobile transaction progresses to states 1, 2, ..., \( N_{L} \), in that order. This is the execution phase and some of the states may be grouped in a request message. The data objects are equiprobably selected from a set of \( DB_{size} \) data objects, and access to a data object is either a 'read/write'(i.e. update) with probability \( P_{r} \) or a read with probability \( 1 - P_{r} \), respectively. At the end of state \( N_{L} \), if successful, the mobile transaction enters into the commit phase at state \( N_{L} + 1 \). Otherwise, the transaction is aborted. The probabilities of abort occurrence for the first run and for any subsequent run are described as \( P_{A} \) and \( P'_{A} \), respectively.

Let \( R \) be the mean response time and \( T_{\text{EXEC}} \) be the mean time of sum of the execution times in states 1, 2, ..., \( N_{L} \), for the ith run of a transaction. During the execution of a mobile transaction, the MH may be disconnected with MSSs. Let \( T_{\text{DISCON}} = T_{\text{DISCON}} \times N_{\text{DISCON}} \) be the mean time of total disconnection during the execution of a mobile transaction, where \( T_{\text{DISCON}} \) is the mean time of a disconnection and \( N_{\text{DISCON}} \) is the mean number of disconnection. In addition, let \( T_{\text{COM}} \) be the mean time of communication for sending requests and receiving the results from MSSs; \( T_{\text{BACKOFF}} \), the mean time from the ith
abort of a transaction until its $(i + 1)$st run is started; and $T_{COMMIT}$, the mean commit time to reflect the updates into the database. Then, the mean response time $R$ can be expressed as follows.

$$R = T_{INIT} + T_{EXEC} + T_{DISCON} + T_{COM} + P_A \times (T_{BACKOFF} + T_{EXEC} + T_{DISCON} + T_{COM} + T_{COMMIT})$$

In the above expression, $T_{INIT}$, $T_{EXEC}$, $T_{DISCON}$, $T_{COM}$, and $T_{COMMIT}$ can be approximated from the hardware queuing model. The remaining quantities to be approximated are $P_A$ and $P_A$.

For an arbitrary mobile transaction $MT$, $\frac{DB_{size}}{C}$ is the probability that at the commit phase the $i$th operation (where $1 < i < N_L$) of $MT$ conflicts with an update from a committing transaction. Let $C = \lambda T_{EXEC}$ be the mean number of committing transactions during the commit phase of $MT$, where $\lambda$ is Poisson arrival rate of transactions per second. The mean number of data objects updated by committing transactions during the commit phase of $MT$ can be expressed as $N_L P_w C$. The probability that in the commit phase the data set of $MT$ does not conflict with any updates of committing transaction during the period of the commit phase can be approximated as $(1 - \frac{DB_{size}}{C}) N_L P_w C$. Using these expressions, the probability of abort $P_A$ is approximated as follows.

$$P_A \equiv 1 - \prod_{i=1}^{N_L} \left(1 - \frac{DB_{size}}{C}\right) N_L P_w C$$

Analogously, $P_A$ can be simply obtained from Equation (2).

5. Performance Evaluation

5.1 System Model

The simulation model for the performance evaluation is essentially a closed queuing model which consists of two distinct components: MHs and MSSs. Each MH is composed of three components: local transaction generator (LTG), local transaction manager (LTM), and cache manager (CM). Each MSS has three components: mobile transaction manager (MTM), replica manager (RM) and data manager (DM). Figure 5 depicts the reference model of the mobile client-server architecture. LTG is responsible for generating a local transaction, which is modeled as a sequence of read/write operations. LTM plays a role of managing the local transaction from beginning to commitment.

MTM takes the responsibility of scheduling transactions by validating their executions according to an optimistic or pessimistic schemes. It also propagates callback messages synchronously/asynchronously in order to notify MHs of data updates. In the case of synchronous propagation, the propagation interval is specified by the parameter, broadcasting_interval. RM maintains mutual-consistency of replicated objects stored at MSSs according to the read-one write-all (ROWA) approach. Since we assume that a database is fully replicated at all MSSs, the number of MSSs, num.MSSs, specifies the number of participating sites in an update process. DM consists of a number of disk servers. The parameter, num.MSS.disks, is used to specify the number of disk servers in DM.

In the simulation model, CPU servers are utilized to perform their jobs by LTM and MTM. The parameters, num.MH.cpu's and num.MSS.cpu's, specify the number of CPU servers in an MH and an MSS, respectively. The CPU servers are modeled as a pool of servers, all identical and serving a common CPU queue in a FIFO manner. The simulation model was programmed in CSIM[17] discrete-event simulation package, and comparative computational experiments have been carried out on a SUN Ultra-SPARC running under Solaris 2.5.

5.2 Simulation Parameters

The simulation parameters are classified into two categories: system parameter, and application parameter. The system parameters are relevant to the system environment under which the simulation study is performed, and the application parameters characterize the transactions to be processed in the system (Table 1). Some of the actual values used for simulation parameters were chosen from [1][6][12] where the justification of the values were fully discussed, and the other values were determined to be reasonable for investigating performance differences of OTS/MT, TO while keeping simulation times reasonable (Table 2). Parameters that vary from experiment to experiment will be given with the description of the relevant experiments.

The size of a database stored at each MSS is as-
Table 1 Simulation Parameters Description

<table>
<thead>
<tr>
<th>System Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu.time</td>
<td>CPU time to accessing page</td>
</tr>
<tr>
<td>mem.size</td>
<td>size of objects in pages</td>
</tr>
<tr>
<td>cache.size</td>
<td>size of MH cache in pages</td>
</tr>
<tr>
<td>memobj</td>
<td>size of a page in bytes</td>
</tr>
<tr>
<td>mem.objpage</td>
<td>mean number of objects per page</td>
</tr>
<tr>
<td>mem.MH.size</td>
<td>number of objects in an MH</td>
</tr>
<tr>
<td>mem.MSS.size</td>
<td>number of objects in an MSS</td>
</tr>
<tr>
<td>mem.objpage</td>
<td>number of objects in a page</td>
</tr>
<tr>
<td>mem.MH.data</td>
<td>number of disks in an MH</td>
</tr>
<tr>
<td>mem.MSS.data</td>
<td>number of disks in an MSS</td>
</tr>
<tr>
<td>comm.cpu.delay</td>
<td>CPU time for accessing page</td>
</tr>
<tr>
<td>comm.MH.data</td>
<td>CPU time for message transfer overhead</td>
</tr>
<tr>
<td>comm.MSS.data</td>
<td>CPU time for message transfer overhead</td>
</tr>
<tr>
<td>comm.bandwidth</td>
<td>broadcasting interval for call back messages</td>
</tr>
<tr>
<td>mem.MH</td>
<td>number of MHs</td>
</tr>
<tr>
<td>mem.MSS</td>
<td>number of MSSs</td>
</tr>
<tr>
<td>mem.SSD</td>
<td>data transfer rate of hard-disk channel</td>
</tr>
<tr>
<td>mem.bandwidth</td>
<td>ratio of a jwalking area to cell area</td>
</tr>
<tr>
<td>mem.availability</td>
<td>ratio of the number of channels to that of MHs</td>
</tr>
<tr>
<td>mem.delay</td>
<td>ratio of a jwalking area to cell area</td>
</tr>
</tbody>
</table>

Table 2 Simulation Parameters Setting

<table>
<thead>
<tr>
<th>System Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>comm.delay</td>
<td>1000</td>
</tr>
<tr>
<td>memsize</td>
<td>1000</td>
</tr>
<tr>
<td>memobj</td>
<td>1000</td>
</tr>
<tr>
<td>mem.objpage</td>
<td>1000</td>
</tr>
<tr>
<td>mem.MH.size</td>
<td>10</td>
</tr>
<tr>
<td>mem.MSS.size</td>
<td>1</td>
</tr>
<tr>
<td>mem.bandwidth</td>
<td>10 MBs</td>
</tr>
<tr>
<td>mem.SSD</td>
<td>180 Kbps</td>
</tr>
<tr>
<td>mem.availability</td>
<td>1</td>
</tr>
<tr>
<td>mem.cpu.delay</td>
<td>35 milliseconds</td>
</tr>
<tr>
<td>comm.cpu.delay</td>
<td>12 milliseconds</td>
</tr>
<tr>
<td>comm.MH.data</td>
<td>10 milliseconds</td>
</tr>
<tr>
<td>comm.MSS.data</td>
<td>10 milliseconds</td>
</tr>
<tr>
<td>comm.bandwidth</td>
<td>10 MBs</td>
</tr>
<tr>
<td>mem.bandwidth</td>
<td>180 Kbps</td>
</tr>
<tr>
<td>mem.availability</td>
<td>0.5</td>
</tr>
</tbody>
</table>

5.3 Performance Indices

The primary performance indices used in our experiments are commit time and restart ratio. The commit time is defined as the average time for performing a commit process. This is measured with the interval between two time instances: (1) a time instance that an MH issues a commit request and (2) a time instance that the MH receives a commit-grant message. The commit time includes the processing time for validating aborted transactions. Thus, it presents performance indices containing the effect of aborted transactions, unlike as the traditional response time. The restart ra-
tio gives the average number that a transaction has to restart per commit. This is computed as the ratio of the number of transaction-restarting event to the number of transaction commits.

An additional performance-related index is used in analyzing the results of our experiments. The index is the transaction throughput. The transaction throughput is defined as the number of transactions successfully completed per a second.

5.4 Simulation Results

5.4.1 Effect of Multiprogramming Level

The purpose of this experiment is to investigate the performance of the two scheduling algorithms, i.e. OTS/MT, TO and OCC, as load of the system varies, and to see how performance is impacted by different level of the multiprogramming.

Figure 6-(a) presents the average commit time of OTS/MT, TO and OCC. As expected, the mean commit time increases with the multiprogramming level gets higher. At lower multiprogramming level (i.e. until the multiprogramming level is 30), OTS/MT exhibits inferior commit time than that of other algorithms. This is due to its optimistic approach of OTS/MT in lower conflict. In a lower data conflict situation, only one data transmission at the commit in OTS/MT. However, in case of TO, it needs a wireless communication in every transaction operation except data is in cache. In case of OCC, since it uses more wireless communication for comparing read/write set with committed transactions, it takes more time for committing a transaction than OTS/MT. Under the multiprogramming level 15, OCC shows inferior commit time than TO. Above the multiprogramming level 30, two optimistic algorithms took more commit time. In this experiment, wireless channel availability is set to 20. If an MH can get a channel immediately, the commit time of OTS/MT is faster than that of other algorithms. Otherwise, TO is inferior in the commit time result. Figure 6-(b) presents the transaction throughput results for each algorithm. The throughput rate of each algorithm initially increases as the multiprogramming level is increased and reaches a peak and then finally decreases. In this case, the thrashing points of OTS/MT, TO and OCC correspond to the multiprogramming level of 20, 30 and 25, respectively. In figure 6-(b), in the range of 10 to thrashing points of the multiprogramming level, the throughput rate curves of each algorithm rapidly increase since there are not enough transactions to keep system resources busy at the range. However, on and after the multiprogramming level of thrashing points, the curve decreases with the increase in the multiprogramming level. This is because increasing the multiprogramming level further only makes data contention higher. Figure 6-(b) also indicates that OTS/MT provides high performance than TO and OCC. After the multiprogramming level of thrashing points, the gap between two algorithms becomes small with the increase in the multiprogramming level. In case of OCC, it exhibits better throughput only near the thrashing point than TO.

The transaction restart ratio of two algorithms is presented in figure 6-(c). The restart ratio curves are similar to that of average transaction commit time, in figure 6-(a). This is due to that the restart ratio has an effect to the average transaction commit time.

5.4.2 Effect of Timeout

This experiment examines the impact of timeout value on the performance of the algorithms. All parameters except multiprogramming level and timeout value are identical to those of 5.4.1. Varying the timeout value provides the impact of the transmission quality of wireless communication. In case of communication failure or delaying for a while, the wireless channel is thought broken or in unstable state. Figure 7-(a) presents the average commit time results obtained by varying timeout value. At low timeout value, i.e. until the timeout value is 10 millisecond, OTS/MT has better performance than TO because TO has more wireless communications and transmission failures by timeout than OTS/MT. As the timeout value increases, there is a crossover in the commit time curves of OTS/MT and TO for the timeout value=10 millisecond. After that value, TO's curve becomes little lower than OTS/MT and slightly decreases. Note that the curve of OTS/MT has almost no effect of timeout values and in stable communication environment, even though OTS/MT takes more commit time. The commit time curve of OCC
is similar slope with OTS/MT although overall performance of OCC is worse. Under the 7 millisecond, OCC is better than TO. This means that optimistic approach is not affected by varying timeout values.

Figure 7-(b) gives the throughput results for the two algorithms in varied timeout values. Like the commit time curves, at the low timeout values, the throughput rate of OTS/MT outperforms that of TO, but as the timeout value is increased the throughput rate of TO becomes to surpass that of OTS/MT. This is due to unstableness of wireless communication. In case of TO, in the range of 8-10 milliseconds of timeout value, the throughput rate rapidly increases and after that timeout value, throughput curves are gradually increased. Like commit time, TO has better performance in the stable communication environment. Further, OCC shows better throughput than TO under lower timeout values and overall throughput is worse than OTS/MT.

Figure 7-(c) shows the restart ratio results for varying the timeout value. As expected, the restart ratio of TO becomes to be superior to those of OTS/MT and OCC with the timeout value increase. The worse restart ratio curve of OCC shows similar slope with OTS/MT. Note that the results of throughput of varying the timeout value are almost identical to those of commit time. The timeout method is needed to decide the communication failure, but it caused frequent transaction abort as the timeout value is decreased and the long-lived transaction with its increase. After all, this introduces the more transaction abort and the lower transaction throughput.

5.4.3 Effect of Write/Read Ratio

The purpose of this experiment is to evaluate the effect of the write/read ratio in one transaction. The write/read ratio percentage implies the degrees of transaction conflict. There are more transaction conflicts as the write/read percentage is increased. Figure 8-(a) shows the commit time results for varying the ratio of write/read operations. At low percentage of write/read, i.e., 10%, the results of OTS/MT and TO algorithms begin with the same commit time. The curve of OCC exhibits similar slope with that of OTS/MT and shows worse commit time. As the write/read ratio is increased, the commit time curve of TO is rapidly increased and that of OTS/MT is raised slightly. The bigger the range of write/read ratio is, the more the difference between two algorithms becomes. This result is due to the optimistic approach of OTS/MT which is preferable in the case of low read/write conflict between operations. Since the write/read ratio may affect the read/write conflict, TO is inferior to OTS/MT as the write/read ratio increases. In case of OCC and OTS/MT, write/read ratio does not effect the gap between curves of two algorithms.

Figure 8-(b) gives the throughput results for three algorithms in varied write/read percentage. In figure 8-(b), in the whole range of write/read ratio, the throughput rate curve of three algorithms decreases since there are more read/write operation conflicts as the write/read percentage increases and OTS/MT shows better performance than TO and OCC.

Figure 8-(c) shows the restart ratio results for varying the percentage of write/read. TO has small restart ratio than OTS/MT in the range of 10% - 15% of write/read percentage. OTS/MT shows small restart
ratio that OCC in the whole range. The curves of OTS/MT and TO are crossed at the point of 15%. After that point, the curve of TO is increased and that of OTS/MT is slightly decreased in the whole range. In case of TO, the read/write conflict check is done with every operation in the transaction. On the other hand, OTS/MT checks once per a transaction in the transaction validation time. Therefore, the restart ratio of TO is easier to be affected by write/read ratio than that of OTS/MT. Further, in case of OTS/MT, the validation of a write operation takes more time than that of a read operation. Since the write operation needs two timestamps for solving read/write conflict which is twice in case of the read operation, the commit time result curve is increased as the restart ratio result curve is decreased in OTS/MT. While OCC shows worst restart ratio, the curve slope of OCC is the same as OTS/MT. Therefore, optimistic approach is less easy to be affected by write/read ratio.

5.4.4 Effect of Bandwidth

The purpose of this experiment is to examine the impact of the bandwidth of wireless communication channel on the performance of the algorithms. Varying the bandwidth speed value provides the impact of the transmission delays in wireless communication. All parameters except bandwidth are identical to those of other experiments. Figure 9-(a) shows the average commit time of OTS/MT, TO and OCC for varying the wireless communication bandwidth. As expected, the average commit times of three algorithms are decreased as bandwidth is increased. After 56,000 bps, the curves of two algorithms are not changed. This means that the communication speed exceeded 56K bps is quite fast for OTS/MT, OCC and TO. In the range below 96,000 bps, OTS/MT and OCC are almost identical. Over 96,000 bps, OTS/MT shows better commit time than OCC. Under the 14,400 bps, TO is superior to OTS/MT. In the range over the 14,400 bps, OTS/MT shows better average commit time than TO. This is due to that OTS/MT have to transfer more data for scheduling a transaction than TO and OCC. For an object which is read/written by an operation in a transaction, OTS/MT is needed to transfer two (one in read operation) timestamps and one result object value of an operation for validating the transaction, i.e. timestamp of object at the time it has been written (read). On the other hand, TO is needed only one timestamp per operation for scheduling when the object is read/written and isn’t needed to data transfer at commit phase. Therefore, the gap between two algorithms is decreased as the bandwidth is increased. In case of OCC, the curve of OCC is the same as that of OTS/MT. OCC takes more commit time as the speed of wireless communication channel becomes fast.

Figure 9-(b) presents the throughput results obtained by varying wireless bandwidth. At low bandwidth range, the throughput rates of three algorithms has no difference. As the bandwidth is increased, the throughput rate curve of OTS/MT is sharply increased. On the other hand, the throughput rate curve of TO is slowly increased. The throughput rate curve of OCC is more slowly increased. In the whole bandwidth range, OTS/MT shows better throughput rate than TO and TO shows better than OCC. OTS/MT has more radical difference according to the various wireless bandwidths than TO and OCC, and has better performance in the range above 14,400 bps. In the results of the throughput rate, OTS/MT has a little superiority in the low bandwidth and has a lot of advantage at the high speed area.

Figure 9-(c) shows the restart ratio results for three algorithms in varied wireless bandwidth. As expected with the results from the average commit time, OTS/MS is inferior to TO in the low bandwidth. OCC exhibits the same restart ratio under the low bandwidth and is more slowly decreased than OTS/MT as bandwidth is increased. This means that OTS/MT, which has more data transmission for validating the transaction than TO, is more affected by the slow speed of wireless channel. Since, OCC has more data than OTS/MT, it could not have benefit as much as OTS/MT.

5.4.5 Effect of Channel Availability

This experiment examines the impact of the channel availability on the performance of the algorithms. The number of MHs is set to 25 and the number of channels is varied from 10 to 35, i.e. the channel availability is varied from 0.4 to 1.4. All other parameters are identical to other experiments. Figure 10-(a) and 10-(b) give
the results of average commit time and the results of average restart ratio obtained by varying channel availability. At low channel availability, i.e., 0.4, OTS/MT exhibits inferior commit time than TO. OCC shows worse results than OTS/MT and TO. As channel availability is increased, three algorithms radically shorten the average commit time. Until the channel availability of 0.6, TO shows better performance than OTS/MT and the difference of commit time between two algorithms is decreased. After this point, the throughput rate curve of TO is slightly decreased and that of OTS/MT is more radically decreased than TO. The trend of restart ratio results obtained by varying wireless channel availability is similar to that of average commit time.

Figure 10-(b) presents the results of throughput rate of three algorithms for varying wireless channel availability. In the whole range, OTS/MT outperforms TO and the difference between two algorithms becomes bigger as the wireless channel availability is increased. Furthermore, after the point of 0.8, the restart ratio curve of OTS/MT is continuously increased and that of TO has almost no change. In case of OCC, it shows better throughput than TO under 0.6. After that point, OCC is inferior to TO although OCC is almost not affected by channel availability. Since OTS/MT has more data to transfer via wireless channel, it is easier to be affected by wireless channel availability.

6. Conclusions

In this paper, we proposed an optimistic timestamp scheduling algorithm called OTS/MT which is applicable to mobile transactions that are substantially long-lived and frequently disconnected with mobile support systems. The major advantages of OTS/MT prevent mobile transactions indefinite postponement and cascading delay which are serious drawbacks of optimistic concurrency control scheme and timestamp ordering scheme in mobile computing environment. Additionally, the proposed OTS/MT algorithm is inherently a deadlock-free scheduling scheme. Deadlock-free property is an attractive one in mobile computing environment in which resources such as battery capacity or communication channels are limited.

We justified more usefulness of OTS/MT than TO and OCC in wireless environment by comparing simulation results. In the overall performance results, the throughput of OTS/MT is better than other algorithms. In case of restart ratio and commit time result, OTS/MT shows superior results running with the lower data contention, the higher communication speed and channel availability. Additionally, OTS/MT shows the uniform result in varying timeout values. This means that OTS/MT is not affected by the communication disconnection differently from TO. Contrary to our expectation, OTS/MT shows the stable results for varying write/read ratio. This means that OTS/MT is not much affected by write/read ratio in our environment of low data contention.

In OTS/MT, however, the restart ratio and the commit time results of the higher data contention, the lower communication speed and the lower channel availability are inferior than TO. This is due to the amount of data transmission through the wireless channel. Therefore, in these conditions, we must consider the trade off between the throughput rate and the overhead of restart, the response delay of an issued transaction.

In case of OCC, the slopes of curve in overall comparison are similar to those of OTS/MT. However, the whole performance is inferior to OTS/MT. This means that both optimistic approaches shows analogous tendency in wireless environment. Since OCC uses more wireless channels, it exhibits worse performance than OTS/MT. If OCC will be able to decrease wireless communication like OTS/MT, the results of both algorithms will be almost the same.

As further work, we will make a performance comparison between OTS/MT and other various mobile concurrency control algorithms. We will make the comparison of analytic models of algorithms which are developed for mobile transaction processing.

References


[3] D. Barbara and T. Imielski, Sleepers and Workaholics:


Jonghyun Lee received his Ph.D degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1996. Since 1986, he has been working for Korea Telecom in the area of network management and multiddatabase system. His current research interests include concurrency control, transaction management, and performance modeling in mobile computing environment. He is a member of ACM and IEEE Computer and Communication Society.

Songchun Moon received his Ph.D in Computer Science from the University of Illinois at Urbana-Champaign in 1985. He has been working for KAIST since then. He was a distinguished scholar at the Hungarian Academy of Science and still serves for EUROMICRO as a director. He has developed a multi-user relational database management system IM, which is the first prototype ever in Korea in 1990 and also a distributed database management system DIME in 1992, which is another first prototype ever in Korea. Currently, he is involved in developing a multi-user object-oriented DBMS GOIM.

MinKyo Lee is a Ph.D. student at the department of management engineering of graduate school of management in KAIST. He received B.S degree in computer science from Yonsei Univ., in 1991. M.S. degree in computer science from KAIST, in 1993. His research interests include concurrency control in mobile database systems, distributed transaction processing, and network.
**Fig. 1** Indefinite Postponement of Transaction Validation in OCC

**Fig. 2** Cascading Delay in TO Scheme Due to Disconnection
Fig. 3 - Mobile Database System Model
Fig. 6  Effect of Multiprogramming Level
Fig. 7 Effect of Timeout
(a) Commit Time

(b) Throughput

(c) Restart Ratio

Fig. 8 Effect of write/read ratio
Fig. 9  Effect of Wireless Bandwidth
Fig. 10 Effect of Channel Availability

(a) Commit Time

(b) Throughput

(c) Restart Ratio