Concurrency Control Mechanism for Nested Transactions in Mobile Environment

Ms. Nyo Nyo Yee
Faculty of Information and Communication Technology
University of Technology (Yatanarpon Cyber City)
Pyin Oo Lwin, Mandalay Division, Myanmar

Ms. Hninn Aye Thant
Faculty of Information and Communication Technology
University of Technology (Yatanarpon Cyber City)
Pyin Oo Lwin, Mandalay Division, Myanmar

ABSTRACT
In mobile environment, mobile host can initiate transactions and that transactions may be executed at mobile host or fixed host. Most of the transactions use in mobile environment is flat transactions. In modern world, most of the applications are complex and long-running. Flat transactions could not work properly in complex and long-running applications. Moreover, flat transactions can be performed only commit or rollback and cannot save intermediate results. If transactions rollback, the whole transaction will be re-started. To solve this problem, we proposed a method that based on closed nested transactions because nested transactions are suited for complex application and can save intermediate result. Proposed system is based on existing Two-Shadow Speculative Concurrency Control (SCC-2S) mechanism that solves concurrency control problem (read-write conflict) for nested transactions and complex application. Proposed system solves the facts that could not solve (write-write conflict) in existing SCC-2S algorithm and also adds Priority Control mechanism to improve the performance of the system and to reduce miss deadlines. This method is intended for Mobile Real-Time Database System (MRTDBS). Concurrency Control will perform at the Fixed Host and the results are returned back to the corresponding Mobile Hosts.

Keywords
Concurrency control, fixed host, flat transaction, real-time database, nested transaction.

1. INTRODUCTION
Mobile Real-time Database System (MRTDBS) provide information to Mobile Host (Mobile User). Primary objective of MRTDBS is to minimize missed deadlines. Mobile host can initiate transactions from anywhere and at anytime. When shared data item is updated by multiple transactions from mobile devices at the same time, Concurrency Control (CC) techniques are required to guarantee timely access and correct results (Consistency). General characteristics of mobile environments like mobility, low bandwidth, limited battery power, limited storage, frequent disconnections etc. makes concurrency control more difficult [10].
Various concurrency control algorithms differ from the time when conflicts are detected, and the way they are resolved. Pessimistic Concurrency Control (PCC) and Optimistic Concurrency Control (OCC) alternatives differ from conflict detection and conflict resolution. PCC locking protocols detect conflicts as soon as they occur and resolve them using blocking, while OCC protocols detect conflicts at transactions commit time and resolve them using restarts.

Speculative Concurrency Control (SCC) algorithms combine the advantages of both PCC and OCC algorithms, and avoid their disadvantages. SCC algorithm similar PCC algorithm in those potentially harmful conflicts is detected as early as possible, and it increases the chances of meeting timing constraints. SCC resembles OCC in that it allows conflicting transactions to proceed concurrently, thus it avoids unnecessary delays to meet timely commitment. SCC allows many shadows for uncommitted transactions. But, SCC-2S allows a maximum of two shadows per uncommitted transaction to exist in the system at any point in time: a primary shadow and a standby shadow [1]. Primary shadow means the original nested transaction query to access shared data. Standby shadow means the copy of the original query that does not contain the portion of the query that the primary shadow is already performed.

The rest of this paper is organized as follows: Section 2 briefly introduce mobile database environment. In section 3, we present our proposed method and proposed system architecture. In section 4, we present mathematical expression of proposed method and Section 5 shows performance analysis for pessimistic concurrency control and our proposed method. Section 6 draws the conclusion.

2. MOBILE DATABASE ENVIRONMENT

Mobile database environment consists of Mobile Host (MH), Fixed Host (FH) and Base Station (BS). The communication of the MH and FH is supported by BS. FH and BS are connected with a wired network. Some MH have Database Management System (DBMS) module to perform database operations. Proposed system architecture contains MHs and FH. FH has database system module to perform database operation and MH does not require having database system module. In mobile environment, MH can process its workload in continuously connected mode or in disconnected mode or in intermittent connected mode [8]. In proposed system architecture, mobile host can live intermittent connected mode and after fixed host had performed database operation, the results are returned back to the corresponding mobile host.
There are three types of data dissemination mode in mobile environments. In Broadcast Mode (push process), On-Demand Mode (pull process) and Hybrid Mode [8]. Our proposed model uses On-Demand Mode.

2.1 Classification of Transactions

There are three types of transactions used in database systems. They are flat transactions, nested transactions, and distributed transactions. All of these transactions have four properties. These properties are Atomicity, Consistency, Isolation (Independence), and Durability (or Permanency).

Flat transactions access a single database and are suitable for simple applications [10]. Nested Transactions are constructed from a set of sub-transactions. Each sub-transaction may also have sub-transactions, and nesting can occur to arbitrary depth. Nesting of transactions can be represented by a transaction tree. Transaction at the root of the tree is called top-level transaction (TL-transaction); others are called sub-transactions. There are two types of nested transaction models, open nested transaction model and closed nested transaction model.

![Open Nested transaction Model](image1)

![Closed Nested transaction Model](image2)
In open nested transaction model, parent transaction does not enforce restriction of the execution, rollback, and commitment of its sub-transactions. The parent transaction only invokes sub-transaction, and the sub-transactions executed independently to each other and to the parent[4]. Open nested transaction has only one parent transaction and the transaction is broken down smaller parts and can perform parallel between sub-transaction. Locks are released before parent transaction. Locks are inherited from parent transactions and perform intra-transaction parallelism [6]. Figure 1 defines open nested transaction model. In closed nested transaction model, locks are released after parent transaction and perform inter-transaction parallelism and can be used for the two or more transaction access the same data item concurrently [3]. Closed nested transaction is especially designed for conflict between one nested transaction and other nested transactions not only between in one transaction. Figure 2 defines closed nested transaction model.

Distributed transactions concern with the division of transactions due to need of accessing distributed resources. Special distributed algorithms are needed to handle locking of data and committing of transactions [9].

2.2 Execution Modes in Mobile Environment
Transactions are initiated at mobile host but may be executed on mobile host or fixed host or the execution may be distributed between mobile host and fixed host respectively. There are five execution modes in Mobile Environment. They are

• Complete Execution on Fixed Network
  Transaction is initiated at mobile host but is completed executed at fixed network. In this approach, mobile host acts as a thin client.

• Complete Execution on MH
  Transactions are initiated at mobile host and are executed on mobile host. This approach requires mobile hosts to have processing and storage capabilities as well as managing data. But, reconciliation is needed with fixed host at some point in time.

• Distributed Execution on MH and Wired Network
  Transaction is initiated at mobile hosts and the execution is distributed among mobile host and fixed host. A sub-transaction is executed at fixed host and another one at mobile host. This approach helps in minimizing the communication between the fixed host and mobile hosts.

• Distributed Execution among several MHs
  Transaction is distributed among several mobile hosts for execution. It provides a peer-peer strategy. A mobile host acts as a server for other
mobile hosts so that the execution is distributed between them. The selection of a mobile host for execution of a transaction is location based.

• Distributed Execution among MHs and FHs

Transaction execution is distributed among several mobile hosts and fixed hosts respectively. In this approach, multiple parties may be involved [7].

3. PROPOSED SYSTEM

In our system, mobile user sends query using an uplink channel (pull process). To process the request, database server in Fixed Host use proposed method Two-Shadow Speculative Concurrency Control (SCC-2S) with Priority that avoids conflict (access the same data) to control concurrent access. To increase the degree of parallelism in the execution of long running transaction is primary objective of nested transaction. This system aim for strict deadline nested transactions to improve inter-transaction concurrency. Mobile Host (MH) can live intermittent connected mode (not need to connect the database server) while Fixed Host (FH) perform database operation. After the database operation had performed, FH returns the result back to corresponding MH. MH does not require having Database System (DBMS) module to perform database operations. So, MH acts as a thin client. Figure 3 define flow diagram for proposed system.
During the transactions run, if conflicts occur between the transactions, SCC-2S with Priority algorithm is used to solve the conflict. When the two transactions encounter (read-write) conflict, the read transaction create the standby shadow (copy the transaction) where the conflict is detected. This standby shadow excludes the part of the transaction that the primary shadow (original transaction) already performed. When the two transactions encounter (write-write) conflict, the transaction with late time creates the standby shadow. But the two transactions encounter (write-write) conflict at the same time, our proposed system uses priority control mechanism to determine which transaction should create the standby shadow. Figure 4 defines proposed system architecture.
3.1 Priority Control Mechanism
In this theory, we define T_{wt1} and T_{wt2} are write lock-requesting transactions. Transaction is defined TF if it access data items available at only one Database Module. Otherwise it is defined TS transaction. If transaction is TF, it is assigned high priority otherwise it is assigned low priority.

Write Lock Conflict (Twt1, Twt2)
Begin
  if Priority(Twt1) > Priority(Twt2)
    Create standby shadow for transaction Twt2
  else if Priority(Twt1) < Priority(Twt2)
    Create standby shadow for transaction Twt1
  Else
    Create standby shadow that have greater access item
  end if
End

3.2 Two Shadow Speculative Concurrency Control (SCC-2S) with Priority
Speculative Concurrency Control (SCC) is especially designed for real-time database applications. It relies on the use of redundancy to ensure that serializable schedules are discovered and adopted as early as possible, that increase in timing commitment of transactions with strict timing constraints. Two-Shadow SCC algorithm (SCC-2S), a member of the SCC-nS family, and minimal use of redundancy. SCC-2S uses at most two shadows for each transaction, primary shadow and standby shadow. Original SCC-2S algorithm solves read/write conflict for concurrent transactions. For our
contribution, we want to solve write/write conflict for nested transaction and also add priority control mechanism for nested transaction. The detailed explanation for SCC-2S is:

Let $T_j$ be any uncommitted transaction in the system. The primary shadow for $T_j$ runs (among all the other transactions with which $T_j$ conflicts) to commit. Therefore, it executes without incurring any blocking delays. At that time, standby shadow for $T_j$, is subjected to blocking and restart. It is kept ready to replace the primary shadow, if replacement is needed.

The SCC-2S algorithm resembles Optimistic Concurrency Control with Broadcast Commit (OCC-BC) algorithm in that primary shadows of transactions continue to execute either until they validate or commit or until they are aborted. The difference is that SCC-2S keeps a standby shadow for each executing transaction to be used if that transaction must abort. The standby shadow is basically a replica of the primary shadow, except that it is blocked at the earliest point where a Read-Write conflict is detected between the transaction it represents and any other uncommitted transaction in the system. If required, the standby shadow is promoted to become the primary shadow, and execution is resumed from the point where potential conflict was discovered [5].

Illustration of SCC-2S works is, shown in Figure 5. The two mobile units MU1 and MU2 access the same data item $x$. MU1 execute Transaction $T_1$ to write data item $x$. MU2 execute Transaction $T_2$ to read data item $x$. Both transactions $T_1$ and $T_2$ start with one primary shadow, namely $T_1^0$ and $T_2^0$ respectively. When $T_2^0$ attempts to read object $x$, a potential conflict is detected. At this point, a standby shadow, $T_2^1$, is created. The primary shadows $T_1^0$ and $T_2^0$ execute without interruption, whereas $T_2^1$ blocks. Later, if $T_1^0$ successfully validates and commits on behalf of transaction $T_1$, the primary shadow $T_2^0$ is aborted and replaced by $T_2^1$, which resumes its execution, we hope to commit before its deadline [2].

![Figure 5. Schedule with a standby shadow promotion](image-url)
It is possible that multiple conflicts develop between executing transactions. The three mobile units MU1, MU2 and MU3 access the same data. MU1 execute Transaction T1 to write data item y. MU2 execute Transaction T2 to read data item x and data item y. MU3 execute Transaction T3 to write data item x. Figure 6 illustrates the behavior of SCC-2S when a second conflict develops between T2 and another transaction T3. In particular, the primary shadow T1₀ of T1 attempts to write an object y that both shadows T2₀ and T2₁ had previously read. In this case, the primary shadow T2₁ create the standby shadow T2₂ to solve conflict with transaction T1.

The SCC-2S algorithm allows at most two shadows for the same transaction to co-exist at any given time. In particular, after T2₁ is promoted to become the primary shadow for T2, a standby shadow T2₂ is forked off to account for the read-write conflict between T2₁ and T1[2].

Figure 6. Schedule with two standby shadows

For our contribution, we add write-write conflict for concurrency control in Figure 7. The two mobile units MU1 and MU2 access the same data item x. MU1 execute transaction T1 to write data item x. MU2 execute Transaction T2 to write data item x. Write conflict time for transaction T2 late. So transaction T2 create standby shadow. For example, Figure 8 defines closed nested transaction with their time. Transaction T1 perform write operation on three database items(a,x,z) and transaction T2 perform write operation on three items (b,x,y). Figure 9 shows the time when using our proposed method.

Figure 7. Schedule with a standby shadow promotion for write-write conflict
But, in Figure 10 write conflicts occur at the same time. At that time, our system uses the priority control mechanism. Assume transaction $T_1$ is TF transaction and transaction $T_2$ is TS transaction. So, the standby shadow $T_2^1$ is created for transaction $T_2$. Figure 11 shows the time when using our proposed method (SCC-2S with priority).

Figure 10. Schedule with a standby shadow promotion for write-write conflict using priority control mechanism
4. MATHEMATICAL EXPRESSION FOR PROPOSED METHOD

Let \( T = T_1, T_2, T_3, \ldots, T_m \) be the set of uncommitted transactions in the system. Let \( T_{\text{primary}} \) and \( T_{\text{standby}} \) be primary and standby shadows executing on behalf of the transaction set \( T \), respectively. For each standby shadow \( T^j_r \) in the system, a set \( \text{WaitFor}(T^j_r) \) is maintained, which contains a list of tuples of the form \((T^s_s, Y)\), where \( T^s_s \in T \) and \( Y \) is an object of the shared database. \((T^s_s, Y) \in \text{WaitFor}(T^j_r)\) implies that \( T^j_r \) must wait for \( T^s_s \) before being allowed to read or write object \( Y \). The notation \((T^s_s, -) \in \text{WaitFor}(T^j_r)\) is used where there exists at least one tuple \((T^s_s, Y) \in \text{WaitFor}(T^j_r)\), for some object \( Y \). Details of the SCC-2S algorithm are defined as follows:

1) When a new transaction \( T_s \) is requested for execution, a primary shadow \( T^0_s \in T_{\text{primary}} \) is created and executed.

2) Whenever a primary shadow \( T^i_s \) wishes to read an object \( Y \) that has been written by another shadow \( T^j_r \), then:
   a) If there is no standby shadow for \( T^s_s \), a new shadow \( T^{i+1}_s \) for \( T^s_s \) is created, such that \( \text{WaitFor}(T^{i+1}_s) = \{(T^r_r, Y)\} \), otherwise.
   b) Let \( T^k_r \) be the standby shadow executing on behalf of \( T^s_s \). If \((T^r_r, Y) \notin \text{WaitFor}(T^k_r)\), then \( \text{WaitFor}(T^k_r) = \text{WaitFor}(T^k_r) \cup \{(T^r_r, Y)\} \).

3) Whenever a primary shadow \( T^i_s \) wishes to write an object \( X \) that has been read by another shadow \( T^j_r \), then:
   a) If there is no standby shadow for \( T^r_r \), then a new shadow \( T^{j+1}_r \) for \( T^r_r \) is created and executed, such that \( \text{WaitFor}(T^{j+1}_r) = \{(T^s_s, X)\} \), otherwise.
   b) Let \( T^k_r \) be the standby shadow executing on behalf of \( T^r_r \). If \((T^s_s, X) \notin \text{WaitFor}(T^k_r)\), then \( T^k_r \) is aborted and a new standby shadow \( T^{k+1}_r \) is started with \( \text{WaitFor}(T^{k+1}_r) = \text{WaitFor}(T^k_r) \cup \{(T^s_s, X)\} \).
4) A standby shadow $T^s_i$ is blocked whenever it wishes to read any object that has been written on behalf of any of the transactions in $\text{WaitFor}(T^s_i)$. 

5) Whenever it is decided to commit a primary shadow $T^s_i$ on behalf of transaction $T^s$, then 

   a) If $(T^s_i, -) \in \text{WaitFor}(T^r_j)$ then the primary shadow of $T^r$ is aborted, $T^r_j$ is promoted to become a primary shadow of $T^r$, and a new backup shadow $T^r_{j+1}$ is forked off $T^r_j$, such that $\text{WaitFor}(T^r_{j+1}) = \text{WaitFor}(T^r_j) - \{(T^s_i, -)\}$. 

   b) Any standby shadow of $T^s$ is aborted[1]. 

Mathematical expression for write-write conflict

6) Whenever a primary shadow $T^r_i$ wishes to write an object $X$ that has been written by another shadow $T^s_j$, if the time of transaction $T^r_i$ write an object $X$ is a little late than the time of transaction $T^r_i$ write an object $X$ then, 

   a) If there is no standby shadow for $T^r_i$, then a new shadow $T^r_{i+1}$ for $T^r_i$ is forked off, such that $\text{WaitFor}(T^r_{i+1}) = \{(T^s_i;X)\}$, otherwise 

   b) Let $T^r_k$ be the standby shadow executing on behalf of $T^r_i$. If $(T^s_i;X) \notin \text{WaitFor}(T^r_k)$, then $T^r_k$ is aborted and a new standby shadow $T^r_{k+1}$ is started with $\text{WaitFor}(T^r_{k+1}) = \text{WaitFor}(T^r_k) \cup \{(T^s_i;X)\}$. 

Mathematical expression for write-write conflict with priority control mechanism

7) Whenever a primary shadow $T^r_i$ wishes to write an object $X$ that has been written by another shadow $T^s_j$, if the two transaction $T^r_i$ and $T^s_j$ write the same data object $X$ at the same time, proposed system use priority control mechanism. We assume transaction $T^s_j$ access only one database module and it is local transaction and transaction $T^r_i$ access more than one database module and it is global transaction. 

   a) If there is no standby shadow for $T^r_i$, then a new shadow $T^r_{i+1}$ for $T^r_i$ is forked off, such that $\text{WaitFor}(T^r_{i+1}) = \{(T^s_i;X)\}$, otherwise 

   b) Let $T^r_k$ be the standby shadow executing on behalf of $T^r_i$. If $(T^s_i;X) \notin \text{WaitFor}(T^r_k)$, then $T^r_k$ is aborted and a new standby shadow $T^r_{k+1}$ is started with $\text{WaitFor}(T^r_{k+1}) = \text{WaitFor}(T^r_k) \cup \{(T^s_i;X)\}$. 

5. PERFORMANCE ANALYSIS

Most of concurrency control method based on Pessimistic and Optimistic concurrency control. But in mobile environment, most of the method based on Optimistic Concurrency Control. In Optimistic Concurrency Control, each transaction perform database operation using three distinct phases- read phase, validation phase and write phase. Moreover, Optimistic Concurrency Control detects conflicts at transaction commit time and resolve them using restarts. In mobile environment, to use Optimistic Concurrency Control
MHs have Database System Module to perform database operations. After finishing database operation, MHs send result back to FH to check conflicts or not. If conflicts occur between transactions, only one MH write request is performed and other MHs must perform database operations again.

Pessimistic concurrency control is based on two phase locking protocol in which a row is unavailable to users from the time the record is fetched until it is updated in the database. So, if conflicts occur between transactions, conflicted transactions perform database operation again. We compare our proposed method with pessimistic concurrency control because of pessimistic concurrency control perform database operation at FH. Figure 12 shows write-write conflicts occurs between transactions $T_1$ and $T_2$. Transaction $T_2$ finish time is 130. Moreover, if transactions use more than one database, this method can increase transaction processing time and can reduce system performance. Figure 13 uses proposed method and Transaction $T_2$ finish time is 110. So, our proposed method can improve system performance and can reduce transaction processing time.

![Figure 12](image1.png)

**Figure 12.** Pessimistic concurrency control method perform operation and the finish time for Transaction $T_2$

![Figure 13](image2.png)

**Figure 13:** Proposed method with write/write conflict with priority
6. CONCLUSIONS
Two Shadow Speculative Concurrency Control (SCC-2S) with Priority is a powerful mechanism for concurrency control in Mobile Real-time Database System (MRTDBS). SCC-2S with Priority provides high respond time and throughput. SCC-2S with priority relies on redundancy to ensure that serializable schedules are discovered and adopted as early as possible, thus increasing the likelihood of the timely commitment of transaction with strict timing constraints. SCC-2S with Priority decreases the number of missed deadlines, reduce battery power and memory usage in the system. In SCC-2S with Priority, shadow transactions execute on behalf of a given uncommitted transaction so as to protect against the hazards of blockages and restarts. Moreover, MHs cannot require to have database system module and MHs can live thin clients.

7. ACKNOWLEDGMENTS
I would like to thank my supervisor, Dr. Hninn Aye Thant, Associate Professor, Faculty of Information and Communication Technology, University of Technology (Yatanarpon Cyber City), Pyin Oo Lwin, Myanmar, for her excellent guidance, where the initial scope of the thesis was defined, and throughout the process of writing this thesis.

REFERENCES
[4] Hamzeh Khazaei, “Mobile Database System”, Math & Computer Science Department, Amirkabir University of Technology, (Tehran Polytechnic), Hamzeh. khazaei@aut.ac.ir