Concurrency Control Mechanism using Two-Shadow Speculative Concurrency Control with Rules for Closed Nested Transactions Model in Mobile Environment

Ms. Nyo Nyo Yee, and Ms. Hninn Aye Thant

Abstract— In mobile environment, mobile host can initiate transactions and that transactions may be executed at Mobile Host (MH) or Fixed Host (FH). Most transactions use in mobile environment is flat transactions. Flat transactions could not work properly in complex and long-running applications and can be performed only commit or rollback and cannot save intermediate results. If transactions rollback, the whole transaction will be restarted. To solve this problem, we proposed a method that based on closed nested transaction model because nested transactions are suited for complex application and can save intermediate result. Proposed method (Two Shadow Speculative Concurrency Control with Rules) solves write-write conflict for closed nested transaction model in mobile environment. Concurrency control perform at FH and the results are returned back to the corresponding MHs. MH does not require to have database system module. Proposed method provides high respond time, throughput and reduces the number of miss deadlines.

Keywords—Concurrency control, fixed host, flat transaction, nested transaction.

I. INTRODUCTION

REAL-TIME Database System (RTDBS) in Mobile Environment provide information to Mobile Host (Mobile User). Primary objective of RTDBS is to minimize missed deadlines. Mobile host(MH) 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.

Various concurrency control algorithms differ from the time when conflicts are detected, and the way they are resolved. Two-Shadow Speculative Concurrency Control with Rules (SCC-2S with Rules) algorithm combines the advantages of both Pessimistic Concurrency Control (PCC) and Optimistic Concurrency Control (OCC) algorithms, and avoid their disadvantages. Proposed method (SCC-2S with Rules) based on Two-Shadow Speculative Concurrency Control (SCC-2S). SCC-2S with Rules 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 compares the outcomes between pessimistic concurrency control and our proposed method. Section 6 draws the conclusion.

II. MOBILE DATABASE ENVIRONMENT

Mobile database environment consists of Mobile Host (MH), Fixed Host (FH) and Base Station (BS). Some MH have Database Management System (DBMS) module to perform database operations. Transactions are initiated at MHs but may be executed on MHs, FHs or the execution may be distributed between MHs and FHs respectively [2]. In proposed system, FH performs database operation and MHs does not require having database system module.

In mobile environment, MHs can process its workload in continuously connected mode or in disconnected mode or in intermittent connected mode [3]. In proposed system architecture, MHs can live intermittent connected mode while FH perform database operation. After FH had performed database operation, the results are returned back to the corresponding MH. There are three types of data dissemination mode in mobile environments. Broadcast Mode
(push process), On-Demand Mode (pull process) and Hybrid Mode. In Broadcast Mode, broadcast server periodically broadcast most popular data on some wireless channel form which users can listen and, if necessary, download if required data. There is no uplink channel involved in this mode. On-Demand Mode allows a client to request specific data which is not available in the current broadcast or may never appear in the broadcast. The client send the query for require data through an uplink channel. In Hybrid Mode, broadcast and on-demand modes combined [4]. Our proposed model use On-Demand Mode.

A. Classification of Transactions

There are three types of transactions used in database management system. They are flat transactions, nested transactions and distributed transactions. All of these transactions have four properties. These properties are Atomicity, Consistency, Isolation and Durability. Atomicity: A transaction is an atomic unit of processing and it is either performed entirely or not at all. Consistency: A transaction's correct execution must take the database from one correct state to another. Isolation/Independence: The updates of a transaction must not be made visible to other transactions until it is committed (solves the temporary update problem). Durability(or Permanency): If a transaction changes the database and is committed, the changes must never be lost because of subsequent failure.

Flat transactions access a single database and adequate for simple applications. The traditional flat transaction will not be powerful enough to handle complex transaction calculation. The main problem of flat transaction is that there is no way to stop the transaction halfway. If the transaction is stop or abort, the whole transaction will be cancelled and restarted from the beginning [5].

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 [6]. There are two types of nested transaction model, open nested transaction model and closed nested transaction model.

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-transactions. Locks are released before parent transaction and perform intra-transaction parallelism. Open nested transactions require compensating transaction to maintain consistency of the database when rollback occurs in sub-transactions. If a lot of conflict occurs, it requires a lot of compensating transactions. “Fig. 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 [7]. Closed nested transaction is especially designed for conflict between one nested transaction and other nested transactions. “Fig. 2” defines closed nested transaction model. In proposed method, we restrict a transaction to have at most one child at a time like linear nesting for closed nested transaction model. Thus the tree of running descendants of any transaction is a line, not a general tree structure.

III. PROPOSED SYSTEM

In our system, mobile user sends query using an uplink channel (pull process). To process the request, database server (include two databases ) in Fixed Host use proposed method Two-Shadow Speculative Concurrency Control (SCC-2S) with Rules 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 aims for closed nested transaction model to improve inter-transaction concurrency. 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. “Fig. 3” defines flow
Fig. 3 Flow diagram for proposed system

A. Two Shadow Speculative Concurrency Control (SCC-2S) with Rules

Two – Shadow Speculative Concurrency Control (SCC-2S) is especially designed for real-time database applications. Two-Shadow SCC with Rules (SCC-2S with Rules) based on SCC-2S. SCC-2S with Rules uses at most two shadows for each transaction, primary shadow and standby shadow. Original SCC-2S algorithm solves read-write conflict for concurrent transactions. We add rules in existing SCC-2S algorithm to solve write-write conflict and aims for closed nested transaction model. 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 standby shadow excludes the part of the transaction that the primary shadow (original transaction) has already performed.

Illustration of how SCC-2S with Rules works is shown in “Fig. 6”. The two mobile hosts MH1 and MH2 access the same data item x. MH1 execute Transaction \( T_1 \) to write data item x. MH2 execute Transaction \( T_2 \) to write 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 write 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 \( T_1 \) 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 [9].

![Fig. 6 Schedule with a standby shadow promotion for write-write conflict](image)

We use two databases to implement my proposed method. “Fig. 7” defines closed nested transaction with their time. MH1 executes Transaction $T_1$ want to go Nay Pyi Taw (NPT) to Brunei Darussalam (BWN). There is no direct flight for NPT to BWN. So, by using transit three nested transaction NPT to Yangon (RGN), RGN to Bangkok-Don Mueang (DMK), DMK to BWN. MH2 executes Transaction $T_2$ want to go Bagan Nyaung-U (NYU) to Bali (DPS). There is no direct flight. So, NYU to RGN, RGN to DMK, DMK to DPS. Conflict occurs in RGN to DMK transit. The reach time for Transaction $T_2$ (MH2) for RGN-DMK transit is later than Transaction $T_1$ (MH1). So, MH2 creates standby shadow.

![Fig 7. Nested transaction with their time](image)

“Fig. 8” shows the time when using our proposed method. But, in “fig. 9”, the two mobile hosts MH1 and MH2 access the same data item at the same time. All the require data for transaction $T_1$ (MH1) can get only one database and the require data for transaction $T_2$ (MH2) need more than one database. So, the standby shadow $T_2^1$ is created for transaction $T_2$. “Fig. 10” shows proposed method with write-write conflict at the same time.

![Fig. 8 Proposed method with write-write conflict at the same time](image)

### B. General Rules for Proposed Method (SCC-2S with Rules)

Use two database for propose method. We use fid for flightId, rId for routeId, date for reserveDate, r for reachtime, n for no of transit count

Begin

Income write lock request transaction $T_{wt}$

Add $T_{wt}$ into object array named list

for(int $i=0;i<list.length-1;i++$) {
  for(int $j=i+1;j<list.length;j++$) {
    If(list[$i$].fId&&rId&&date ≠ list[$j$].fId&&rId&&date)
      transaction run concurrently.
    Else if (list[$i$].fId&&rId&&date == list[$j$].fId&&rId&&date)
      If (list[$i$].fId&&rId.r < list[$j$].fId&&rId.r)
        Create standby shadow for list[$j$]
      Else if (list[$i$].fId&&rId.r == list[$j$].fId&&rId.r)
        Check data can get only one database or not
        If (list[$i$] access only one database and list[$j$] access more than one database)
          Create standby shadow for list[$j$]
        Else if (list[$i$] and list[$j$] access only one database)
          Check n
          If ($n$ of list[$i$]> $n$ of list[$j$])
            Create standby shadow for List[$i$]
          Else
            Create standby shadow for List[$j$]
        Else if (list[$i$] and list[$j$] access more than one database)
          Check n
          If ($n$ of list[$i$]> $n$ of list[$j$])
            Create standby shadow for List[$i$]
          Else
            Create standby shadow for List[$j$]
      Else if (list[$i$] and list[$j$] access more than one database)
        Check n
        If ($n$ of list[$i$]> $n$ of list[$j$])
          Create standby shadow for List[$i$]
        Else
          Create standby shadow for List[$j$]
      Else
        Create standby shadow for List[$i$]
    Else
      Create standby shadow for List[$i$]
  }
}

End
Create standby shadow for list[i]
End if
}
}
End

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

1) New transaction \( T_s \) is requested for execution, a primary shadow \( T^{i,0}_s \) in \( T^{primary} \) is created and executed[10].

Mathematical expression for write/write conflict

2) Whenever a primary shadow \( T^i_s \) wishes to write an object \( X \) that has been written by another shadow \( T^j_r \), if the time of transaction \( T^i_s \) write an object \( X \) is a little late than the time of transaction \( T^j_r \) write an object \( X \), then,

a) If there is no standby shadow for \( T^j_r \), then a new shadow \( T^{i+1}_r \) is created, such that \( \text{WaitFor}(T^{i+1}_r) = \{(T_s, X)\} \), otherwise

b) Let \( T^{k+1}_r \) be the standby shadow executing on behalf of \( T^j_r \). If \((T_s, X) \notin \text{WaitFor}(T^{k+1}_r)\), then \( T^{k+1}_r \) is aborted and a new standby shadow \( T^{i+1}_r \) is started with \( \text{WaitFor}(T^{i+1}_r) = \text{WaitFor}(T^{k+1}_r) \cup \{(T_s, X)\} \).

Mathematical expression for write/write conflict at the same time

3) Whenever a primary shadow \( T^i_s \) wishes to write an object \( X \) that has been written by another shadow \( T^j_s \), if the two transaction \( T^i_s \) and \( T^j_s \) write the same data object \( X \) at the same time. We assume transaction \( T^i_s \) access only one database module and transaction \( T^j_s \) access more than one database module.

a) If there is no standby shadow for \( T^j_s \), then a new shadow \( T^{i+1}_s \) for \( T^i_s \) is created, such that \( \text{WaitFor}(T^{i+1}_s) = \{(T_s, X)\} \), otherwise

b) Let \( T^{k+1}_s \) be the standby shadow executing on behalf of \( T^j_s \). If \((T_s, X) \notin \text{WaitFor}(T^{k+1}_s)\), then \( T^{k+1}_s \) is aborted and a new standby shadow \( T^{i+1}_s \) is started with \( \text{WaitFor}(T^{i+1}_s) = \text{WaitFor}(T^{k+1}_s) \cup \{(T_s, X)\} \).

4) A standby shadow \( T^i_r \) is blocked whenever it wishes to write any object that has been written on behalf of any of the transactions in \( \text{WaitFor}(T^i_r) \).

5) Whenever it is decided to commit a primary shadow \( T^{i}_s \) on behalf of transaction \( T^i_s \), then

a) If \((T^i_s, -) \in \text{WaitFor}(T^j_r)\) then the primary shadow of \( T^i_r \) is aborted, \( T^{i+1}_r \) is promoted to become a primary shadow of \( T^i_r \).

b) Any standby shadow of \( T^i_s \) is aborted if exists.

V. SYSTEM OUTCOMES

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.

![Fig. 11 Pessimistic concurrency control method perform operation and the finish time for Transaction T2](image-url)
Fig. 12 Performance Metric for Number of Sub-transaction count

Fig. 13 Performance Metric for Number of Concurrent User

VI. CONCLUSION

Most concurrency control method used for real-time database system in mobile environment solves consistency issue (concurrent access) in flat transactions. Flat transaction could not work in complex and long-running applications and could not save intermediate result. If transactions rollback, the whole transaction will be restarted. So, we proposed Two Shadow Speculative Concurrency Control (SCC-2S) with Rules for concurrency control in Real-time Database System (RTDBS) for Mobile Environment. SCC-2S with Rules provides high respond time and throughput. SCC-2S with Rules decreases the number of missed deadlines, reduce battery power and memory usage in the system. Moreover, MHs cannot require to have database system module and MHs can live thin clients.

ACKNOWLEDGMENT

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 Township, Mandalay Division, 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 (ppt)


