Evolving a model of transaction management with embedded concurrency control for mobile database systems

Subhash Bhalla*

Database Systems Laboratory, The University of Aizu, Aizu-Wakamatsu, Fukushima 965-8580, Japan

Received 31 October 2001; revised 21 July 2002; accepted 1 March 2003

Abstract

Transactions within a mobile database management system face many restrictions. These cannot afford unlimited delays or participate in multiple retry attempts for execution. The proposed embedded concurrency control (ECC) techniques provide support on three counts, namely—to enhance concurrency, to overcome problems due to heterogeneity, and to allocate priority to transactions that originate from mobile hosts. These proposed ECC techniques can be used to enhance the server capabilities within a mobile database management system. Adoption of the techniques can be beneficial in general, and for other special cases of transaction management in distributed real-time database management systems. The proposed model can be applied to other similar problems related to synchronization, such as the generation of a backup copy of an operational database system.

© 2003 Elsevier Science B.V. All rights reserved.

Keywords: Distributed algorithms; Mobile database systems; Non-blocking protocols; Parallel algorithms; Serializability

1. Introduction

Transaction updates by mobile clients are a desirable feature of many applications [26,16]. Most existing research efforts consider a limited case, of the read-only support for mobile clients. Few other studies consider relaxing the criteria of serializability for processing database update requests. Some more studies propose a prolonged execution sequence. In a disconnection prone system, prolonged execution of transactions is undesirable [17,26].

We consider, an environment based on transaction classification. The transactions at the server end are considered to be short and these can be easily restarted on account of few failures. The mobile client’s transactions on the other hand are considered instant execution requests of highest (real-time) priority. The server is assumed to have a high capacity and receives a few cases of mobile client update requests. In many cases, the transaction processing system can execute a mobile client or mobile host (MH) update, with little or no overheads. In the study, conflicts among two mobile client transactions are separately discussed at the end for sake of simplicity.

In order to preserve serializability, the conventional systems depend on 2 phase locking (2PL) protocol [5]. Whereas the 2PL protocol enforces a two phase disciple, the criteria of serializability does not dictate the order in which a collection of conflicting transactions need to execute [5]. This option provides an opportunity to make a modified system that follows 2PL protocol at the transaction manager’s (TM) level, but can be flexible at the data manager’s (DM’s) level. It can permit a interference free and ‘non-blocked’ execution for MH transactions. This change necessitates maintaining ‘lock table’ in the form of site level graphs. Although this is the first effort (to the best of our knowledge) to use the technique for mobile databases, many graph based techniques have been studied earlier by Eich and Garard and Reddy and Bhalla [15,24].

It is proposed to execute a mobile host update (MHU) transaction in a special priority fashion. It may need to wait for another low-priority transaction, only if, that transaction has completed and local DM is participating in the second phase of a two phase commit.

The introduction of these possibilities integrates well with the existing transaction execution models. Earlier efforts at separating read-only transactions and update transactions exist [5,13,14]. The present study is an effort that proposes an implementation strategy for isolation of...
Serializable MHU transactions, for such an execution, that is free from interference by other transactions (Fig. 1).

The contents of this paper are organized as follows. Section 2 describes the background of the proposed approach. In Section 3, a model of the system has been presented. A stochastic process model of resource allocation of data resources has been presented, in Section 4. Based on the inferences provided by the studied model, Section 5 considers adaptation of the results for developing strategies for transaction management in mobile database systems. Finally Section 6 presents summary and conclusion.

2. Background

It is common for designers to extend the available approaches for concurrency control for use within the new system environments. However, we propose to study an analytical model and consider introduction of parallelism.

There have been some efforts at introducing parallelism within the concurrency control function. Earlier proposals attempt to eliminate interference between two classes of transactions. For example, processing Read-only transactions separately by using older versions of data, eliminate interference. Within the new classes, transactions are processed with no interference from each other’s transactions. These can be considered to be executing in parallel.

We propose to study the process of data allocation to executing transactions by using a stochastic process model. The model helps us in examining the parallel activity introduced by the use of classification of transactions. It also provides new insights that can lead to efficient processing of time-critical transactions. In the new environment, the time-critical transactions aim to execute with no interference from the ordinary transactions (Fig. 2). In this light, the characteristics of the 2 Phase Locking based Concurrency Control scheme have been examined, within framework of a Real-Time (time-critical) database system.

3. The system model

Based on the models of 2 phase locking and real-time computational environment with no slack time \([12,18]\), a set of assumptions for executing transactions are organized. It is assumed that a 2 phase locking discipline is followed and the transaction execution is based on the criteria of serializability. Ideally, the MHU transactions should be able to do the following:

- a critical transaction may proceed without interference from other transactions.
- Over ride conventional delays during execution
- integrate with existing modes of transaction executions. The two phases within the 2PL protocol must execute with no blocking;
- execute and commit, i.e. if phase 1 is completed, then phase 2 needs to be completed.

In the following section, a scheme to execute transactions as per a precedence order is described.

3.1. Definitions: mobile database system

Mobile database system (MDS) consists of a set of data items (say set ‘D’). The MDS is assumed to be based on a collection of (fixed) servers that are occasionally accessed by mobile hosts. Our assumptions are similar to earlier examples \([3,26]\). Each site supports a TM and a DM. The TM supervises the execution of the transactions. The DMs manage individual databases. Each mobile host supports a TM, that interacts with a fixed Mobile System Support (MSS) station. That performs other TM functions of interaction with other DMs. The network is assumed to
detect failures, as and when these occur. When a site fails, it simply stops running and other sites detect this fact.

3.2. The transaction model

We define a transaction as a set of atomic operations on data items. The system contains a mixture of instant priority real-time transactions (MHU, or MH reads) and ordinary transactions. We assume that the ordinary transactions can be aborted, in case of a data conflict with the real-time transactions.

The use of real-time database systems is growing in many application areas such as, industrial process control systems, and other time-critical applications. Many approaches for implementation of Real-Time systems are being studied [22,2]. In the real-time system environment, a critical transaction (computational task) is characterized by its computation time and a completion deadline. These systems are characterized by stringent deadlines, and high reliability requirements.

3.3. Problem description

A requirement often imposed on transaction processing systems is that the schedule formed by the transactions, be serializable [5]. A common method of enforcing serializability is based on two-phase locking, i.e. a transaction must get a lock grant for a data item before it proceeds to access it. In a time-critical (mobile database) environment, a locking based approach may render a substantial portion of database inaccessible to an arriving MH transaction.

3.4. Example of a transaction

Consider a database system that uses two-phase locking. Assume that it has data items x1, x2, ..., x6, and transactions T1–T6. Transactions T1, T2, T3 and T6 are read only transactions and use shared locks for items requested, as T1(x2, x4), T2(x1, x2), T3(x1, x2, x4), and T6(x3). Transactions T4 and T5 are update transactions and require exclusive access to data items requested as T4(x1, x2) and T5(x3, x5, x6). The transaction execution per the above requests is shown in Fig. 3. The allocation of data items is shown by ‘+’ for read only transactions (‘w’ indicates wait state). Similarly, ‘X’ allocation of data items in the case of update transactions (‘W’ indicates wait state).

The transaction T4 fails to get data items requested (with exclusive access). Also, the transaction T6 fails to get data items requested (with shared access). Thus, an incoming mobile host transaction (MHT) may need to wait until the executing transactions, complete. This introduces an element of unpredictable delay for incoming MHTs.

<table>
<thead>
<tr>
<th>Item</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3. Allocation of data to incoming transactions.

4. Model of transaction execution

A database is a set of N objects (data items), namely x1, x2, ..., xN. A transaction is a sequence of steps, including read and write actions on the database. A database management system executes a transaction by executing its steps, in the given sequence.

A. Two-phase locking protocol (2PL). In this protocol, a transaction must issue a lock (xi) request before performing a read or write action on object xi. Each transaction is required to do all locking of objects, before any unlocking. This guarantees serializability.

Thus, a lock is released for access by other transactions, in the second phase through an unlock (xi) request. In contrast, during the first phase, the transaction progressively performs locking of objects required for processing.

B. Transaction processing activity. Our main objective is to identify a set of key parameters, which are important from the point of view of availability of data resources for processing MHTs.

For the purpose of making a stochastic process model, consider the duration of time, when the system starts receiving transactions, and processes them up to the time, until one of the transactions, returns locks held by it. In the given model, there are two types of transactions, the executing transactions and the waiting transactions. The objective of the exercise can be stated as:

1. Among the transactions, increase the proportion of the executing transactions; and
2. For the waiting transactions, reduce the waiting time of individual transactions to within finite (deterministic) value.

We are able to relate the process that we have described above to the pure birth process [11]. When exactly n locks have been granted, i.e. n items from the database are held by transactions which have so far arrived at the system, the system is in state n. Further, a stochastic process is a
time-homogeneous birth process, if the probability of a transition from state $n$ to state $n+1$ is given by

$$P(n \rightarrow n+1 \text{ in } (t, t+\Delta t)) = \lambda_n \Delta t$$

where $\lambda_n \geq 0$, and if no other type of transition (e.g., $n \rightarrow n-1$) is possible. It is the rate at which data items are allocated. In order to have the identification of the two processes to be complete, some underlying assumptions are:

1. The duration of the time under consideration for the stochastic model is the time before any transaction completes its processing and returns its locks, since the start of the process;
2. There is a certain fixed probability of a transaction arriving at the system and requiring at least one item;
3. The time interval ($\Delta t$), is sufficiently small, such that at most one item is picked up during such an interval; and
4. The total number of objects ($N$), which constitute the database is very large.

In order to complete the identification of the pure birth process with the above process, we proceed to express $\lambda_n$ in terms of parameters of the model. $\lambda_n$ denotes the rate at which the system may change its state from $n$ to $n+1$. Given $\lambda_n$, the various results that may be desired for a pure birth process model can be obtained. In order to understand the behavior of the transactions a bit more closely, we consider $\lambda_n$ to consist of two components. For any change of state of system from $(n \rightarrow n+1)$, the transaction in question may belong to a group that will find all locks free and go for execution. Or, may belong to the second group of transactions. That is, the transactions that will need to wait for obtaining a lock for some other data item (considering the given duration of time). Mathematically,

$$\lambda_n = \lambda_e + \lambda_w \quad (n = 0, 1, \ldots, N - 1)$$

(1)

where $\lambda_e$ is a component of $\lambda_n$, when the transaction in question belongs to transactions of no wait group. $\lambda_w$ is the component of $\lambda_n$, when the transaction may belong to the wait group. The expression for $\lambda_e$ and $\lambda_w$ are given as (Appendix A):

$$\lambda_e = (N-n)\alpha \Delta t \quad (n = 0, 1, \ldots, N - 1)$$

(2)

$$\lambda_w = n(N-n)\beta \Delta t \quad (n = 0, 1, \ldots, N - 1)$$

(3)

Let $X(t)$ be the number of items, which are in busy state at time $t$, and let $P_n(t)$ is the probability, that at time $t$, there are $n$ items in the resource list, that have been allotted to various transactions. Under the Markov assumptions, $X(t)$ is a pure birth process with parameters $\lambda_n \quad (n = 0, 1, \ldots, N - 1)$. Hence, for such a case [11],

$$P_n(t) = \sum_{j=0}^{n} A_n^j \exp(-\lambda_j t)$$

where

$$A_n^j = \frac{\lambda_0 \lambda_1 \ldots \lambda_{n-1}}{\prod_{i=0,i\neq j}^{n} (\lambda_i - \lambda_j)} \quad (n = 0, 1, \ldots, N - 1)$$

(4)

also $\lambda_i \neq \lambda_j$ for all $i \neq j$.

4.1. Analysis of transaction execution

The expression for $P_n(t)$ is mainly dependent on $\lambda_n$. If a total of $m$ transactions are assumed to have reached the database system, at an instant of time, and the total number of items desired by them are $M$, then $n$ will tend to approach $M$, if each $\lambda_n \quad (n = 0, \ldots, N - 1)$ has a higher value. Apparently, increasing $\lambda_n$ values is beyond our control.

In line with our objective, we consider the components of $\lambda_n$ based on transaction classification, as defined in Eq. (1). We can manipulate the value of one component of $\lambda_n$ at the cost of the other. An increase in the number of locks held by the wait group of transactions can be minimized, in order to increase the number of items free for allotment. This can generate increased number of executions in parallel. In order to increase the contribution to $\lambda_n$ from the transactions in no-wait group ($\lambda_e$) and decrease ($\lambda_w$), we define the term parallelism gain factor (PGF). The value of PGF needs to be a high value.

$$\text{PGF} = \frac{\lambda_e}{\lambda_w}$$

From Eqs. (2) and (3)

$$\text{PGF} = \frac{(N-n)\alpha}{n(N-n)\beta} = \frac{\alpha}{n\beta}$$

(4)

Hence, the two parameters that have a bearing on the performance of transactions (for each unit duration under consideration) are $n$ and $\alpha/\beta$ (It is referred to as a coefficient of non-interference (CNI)).

4.2. Adoption of the result

We consider various ways of obtaining a higher PGF value by varying CNI and $n$. The parameter CNI also varies with $n$. For example, if $n \rightarrow 0$, the values of $\alpha \rightarrow 1$, $\beta \rightarrow 0$. As initially, the arriving transaction would tend to find the items it needs and blockings from other transactions are low. Similarly, considering the end conditions,

- As $n \rightarrow 0$; PGF $\rightarrow$ high values. Intuitively also, the early transactions on entering find most data items and are likely to execute,
• \( \beta \rightarrow 0 \), CNI \( \rightarrow \) maximum values, PGF \( \rightarrow \) high values. It indicates the case of low interference from executing transactions. These have no overlap of data resources with respect to an arriving transaction.

• \( \alpha \rightarrow 1 \), CNI \( \rightarrow \) maximum values, PGF \( \rightarrow \) high values. It indicates the case of those transactions that are likely to find all items free. That is the arriving transaction has no conflict with the executing transaction.

Similarly, as \( n \rightarrow N \); PGF \( \rightarrow 0 \), hence, \( \alpha \rightarrow 0 \), \( \beta \rightarrow 1 \), and CNI \( \rightarrow 0 \). That is, if most of the database items are locked, the arriving transaction will get into a conflict. Executing transactions will block other transactions. Ideally, schemes that can keep \( n \) as low as possible, are good proposals. It supports the known intuitive notions that,

• (Lowering \( \beta \)) Locks must be released quickly and not held unless needed. To achieve these objectives, designers look for fast processors that will tend to execute and finish quickly, thereby avoiding many conflicts; a number of modifications to locking techniques, incorporate this inference, such as, preference for dynamic locking over static locking, and a preference for fine grained locking.

• (Lowering value of \( n \)) if there is few or no transaction in the system, then it is likely that the incoming transaction will obtain the locks that it requires.

• (Lowering \( \alpha \)) An example is that of providing the older version of data to read-only transactions. By use of earlier version for Read-only transactions, these avoid delays on account of executing update transactions.

It follows from considerations of CNI, or \( \alpha \) and \( \beta \), that

• if there are executing transactions, but these have a low or nil overlap with the data item requirements of the incoming transaction, then the incoming transaction will also form part of the no-wait group; that is—techniques that can detect non-interfering transactions can schedule more transactions in parallel; and

• all techniques that increase the CNI, tend to increase the potential of parallel execution with in the conflicting environment.

Table 1: Improving parallelism in the execution of transactions

<table>
<thead>
<tr>
<th>No</th>
<th>Parameter</th>
<th>Case example</th>
<th>Interpretations</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \beta = 0, \alpha = 1 )</td>
<td>Executing read-only transactions in shared locking mode</td>
<td>Parallelism in transaction execute</td>
<td>All transactions execute</td>
</tr>
<tr>
<td>2</td>
<td>( \beta = 1, \alpha = 0 )</td>
<td>Executing two conflicting transactions with all data items in conflict</td>
<td>1. One transaction is blocked</td>
<td>Static locking can provide, execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. Both are in a deadlock</td>
<td>Precedence to one of the transactions</td>
</tr>
</tbody>
</table>

Based on these propositions, we consider, concurrency control approaches that reduces the contention for data items and consequently increase the potential for parallel execution of transactions in the following section.

4.3. Validation of the inferences from the model

We consider the inferences from the above model, for two type of considerations. In the first case, consider, increase in parallelism in general due to selective allocation of data resources. In the second case, we consider, parallel execution of two classes of transactions. These considerations are summarized in Tables 1 and 2. In the following section, we consider a few possibilities for processing time-critical transactions.

5. Transaction processing environment

The above model of transaction execution suggests that as per the ideal situation, the value of \( n \) should be 0. For this purpose, the concurrency control mechanism must be able to ignore any executing ordinary transactions (OTs) that may hold data items required by the MHT. In order to incorporate the situation as a reality, the transaction manager must support transaction classification. These must also have a mechanism to accord priority for the critical transactions such as MHTs with no abort, and
Table 3
Inferences for parallel execution of time-critical transactions

<table>
<thead>
<tr>
<th>No</th>
<th>Parameter</th>
<th>Strategy for critical transaction management</th>
<th>Possible results</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\beta = 0$, low</td>
<td>1) Scheduling known non-interfering transactions 2) Not scheduling ordinary transactions in the presence of critical transactions</td>
<td>Increased Parallelism</td>
<td>Described in Section 5.1</td>
</tr>
<tr>
<td>2</td>
<td>$n = \text{low}$</td>
<td>Embedded concurrency control [7,8]</td>
<td>Isolation of critical MH transactions</td>
<td>(Section 5.2)</td>
</tr>
<tr>
<td>3</td>
<td>$\alpha = \text{high}$, $\beta = 0$, low</td>
<td>(a) High priority transactions acquire locks at start (b) MHTs execute with no interference from long read-only transactions [9]</td>
<td>Precedence to high priority MHTs</td>
<td>(Sections 5.3 and 5.4)</td>
</tr>
</tbody>
</table>

5.1. Scheduling: simultaneous arrival of MHTs

Ideally, one MHT must be in progress on a uniprocessor system. The next MHT in the queue must be started on completion of the first. In this way, the earliest time to completion of $p$ (simultaneous arrival) MHTs will be

$$T_p = \sum_{q=1}^{p} \text{(time of completion of \(MHT_q\))}$$

In case of simultaneous scheduling of multiple MHTs on a multi-programmed system, the earliest time to complete $p$ transactions can be estimated intuitively, as

$$T_s \approx T_p$$

The increase can be attributed to data item blocking delays and overheads due to restart after a deadlock, or other restarts [1]. Thus, as a general scheduling strategy, transactions that have no data items in common should be scheduled in preference over transactions that introduce overlap in data items requested. Also, in the proposed environments that support transaction classification, it may be desirable to not schedule ordinary transactions, during the presence of time-critical transactions. These efforts introduce low values for $\beta$. Similar trends in not scheduling transactions that may get into a conflict have been studied in Ref. [6].

5.2. Embedded concurrency control for mobile DBMSs

It is proposed to execute a MHT in isolation. It permits the MHT to proceed by locking data items. This step
Procedure Validate (OT);
Valid := true;
For each X ∈ Read-set (OT) do
  if X ∈ write-set (Transactions committed after read by OT)
    then Valid := false, exit loop; end;
  if X ∈ (Locked items list of MHTs )
    then wait for release and Valid := false, exit loop;
end;
If valid
  then for each X ∈ write-set (T) do
     < Allot a commit sequence number to T >
     < Commit write-set (T) to database >
  else restart (T);
end.
Fig. 5. Validation procedure for Embedded Concurrency Control (ECC).

reduces the value of $n$ as the domain of locked items is confined to MHTs only. The norm for processing other transactions is based on an additional validation check, as per the criteria of serializability. For this purpose, each transaction is validated before commit.

The validation test for other transactions (OTs) uses the following criteria:

1. (normal) No data item, read by the transaction (OT), has been updated by a transaction after the read, that is—
   Read-set (OT) ∩ Write-set (more recently committed transactions); and
2. (additional) No data item read by the transaction (OT), is in the locked item list of executing MHTs, that is—Read-set (OT) ∩ Locked-items (MHTs).

The transactions that fail to meet the first criteria are aborted, and restarted. The transactions that fail to meet the second criteria can be made to delay commit, so as to let the executing MHT complete its execution. The algorithm for performing, the validation check is given below (Fig. 4).

The possibility of repeated rollback of an ordinary transaction can be eliminated (It can be submitted as a low priority MHT). It can be observed that, such an execution introduces a non-interference environment for a MHT.

The above test is strictly conflict based and OTs need not perform any locking and can be made completely dependent upon a validation test [8].

5.2.1. Proof of correctness

The mobile host update transactions execute as per the criteria of serializability by virtue of the 2PL protocol [5]. As the MHTs completely ignore the presence of OTs, these transactions are executed as per the notion of optimistic concurrency control, with an enhanced validation check. The validation check ensures that an OT is serializable with respect to,

1. the previously committed transactions, and
2. the executing mobile host transactions.

5.2.2. Performance considerations

A drawback associated with adoption of validation based approaches is the possibility of repeated rollbacks. However, the proposed scheme can prevent these rollbacks by resubmitting a rejected transaction as a low priority MHT. This will make the OT execute as per the 2PL protocol and prevent the repeated rollbacks.

Although, in this approach the OTs face an enhanced level of validation, there are three positive aspects that are associated with our proposal. First, the OTs avoid the problem of repeated rollbacks. Also, such a mixed mode of execution enhances the overall level of concurrency, because unlike locking based approaches the validation check is based on testing conflicts using the read-set of the committing transaction. Finally, the main item under focus concerning the performance is the execution of MHT. Although the overall level of concurrency is expected to be improved, the performance of MHTs is enhanced as these execute with no interference with the other transactions within the system (Fig. 5). A performance evaluation study has been presented in Ref. [10].

5.2.3. Comparison with existing approaches

The introduction of this possibility integrates well with the existing transaction execution models. Earlier efforts at separating read-only transactions and update transactions exist [5,14]. The present study proposes an implementation strategy for isolation of mobile host transactions (MHTs), for execution, that are free from interference by other transactions.

A similar approach proposed earlier allows segregation of real-time transactions (RTTs) from other transactions [7]. It is based on transaction classification (Fig. 6). It permits RTTs to overlook locks held by OTs. Another technique called Isolation Only Transactions (IOT) has been proposed in the case of mobile database systems (that suffer from excessive delays on account of disconnections) [19]. This class of transactions is guaranteed to be serializable with all previously committed transactions. For this purpose, each transaction is validated before commit. If a transaction fails the validation phase, it enters a conflict resolution phase. Thus, these transactions perform an additional validation at the end to ensure a conflict free commit. In contrast, in the proposed approach, makes the OTs face an additional validation check, to prevent the conflict from occurring.

Fig. 6. Embedded concurrency control for Transaction Management based on transaction classification.
5.3. Two conflicting mobile transactions

Introduction of different levels of priority (preference in sequence) can be adopted, as in Ref. [1], in which a higher priority transaction, in case of a conflict introduces restart for the lower priority transaction. This achieves the objective of providing the high priority transaction with sure execution without total blocking (α → 1, and β → lower value). The technique reduces the size of the blocked domain of objects, and permits more parallelism. However, it introduces additional roll-back overheads for the lower priority transaction.

Consider the case of two transactions that are bound to be in conflict, in a system. Assume that, there is a complete overlap of data item requirements. In order to reduce the possible number of wait group transactions from two (considering possibility of a deadlock) to one, it is preferable to allow one transaction to proceed with locks on its entire domain of data item requirements (α → 1, β → 0). Hence, a high priority transaction with no slack time left must acquire all its data contents, if the other conflicting transaction has sufficient slack time left. This procedure eliminates few blockings and unnecessary overheads.

Thus, given a pair of conflicting MHTs, it is valuable to allow a mixture of locking techniques. For example, the first transaction may use dynamic locking, the subsequent (especially, transactions that need few items, or are short) transactions could use static locking if, possible. This procedure will prevent the possibility of a deadlock between a conflicting pair of transactions, and will contribute to a lower value of parameter n.

An algorithm based on the above conceptual model has been presented in Ref. [8]. Many graph based concurrency control techniques [15,23,24] can benefit from the precedence interchange possibilities as high lighted by the proposed model [25].

5.4. Isolation from long read-only transactions

Taking frequent backup is an essential part of database operations. Many applications require reading an entire database. Recently proposed algorithms [4,20,21] tend to introduce delays for other executing transactions including the MHTs. A global-read (with no concurrency control) produces an inconsistent database version, in the presence of update transaction execution. A scheme for generation of a backup copy of an operational database has been proposed. In a proposal based on transaction isolation based on the proposed model, it is possible to execute the mobile update transactions (MHTs) in parallel [9]. Update transactions are processed as per the normal two-phase locking criterion. The long-read transaction is made to read an inconsistent (blind) copy of the data with no blocking. In a second phase, the recovery logs (marked log entries) are used to eliminate the data inconsistencies (Fig. 7). The phase 2 is carried out in an off-line fashion.

6. Summary and conclusion

In a mobile database system, transactions need to broadcast data updates. Several server level enhancements are essential. A possibility is demonstrated by considering the mobile client, as a host issuing update (or read) transactions. This class of transactions can be executed as an instant priority real-time transaction with no slack time available. By adopting transaction classification, many changes can be accommodated within 2PL at a low cost that enable the database updates by mobile clients.

A use of an instant priority based execution scheme, based on transaction classification, is necessary to reduce delays due to possibility of resource conflicts with ordinary transactions. In this study, we investigate a procedure that can perform critical functions in parallel to process mobile client’s updates as time-critical updates requiring instant priority. Our study is based on study of the allocation of data resources to a class of transactions (MHTs). The parameters identified by the model are helpful in creating an improved understanding about the transaction processing systems. Based on the inferences from the stochastic process model, a model of transaction execution based on embedded concurrency control (ECC) has been presented. The presented model is successful in isolating the MHTs and permits the execution of a mobile host update transaction without any interference from other executing transactions.
Appendix A

It is assumed that the database consists of $N$ objects. For every change of state ($n \rightarrow n+1$), there is an unassigned object being picked up by the system from the database. In order to have the identification of the two processes to be complete, some underlying assumptions are:

1. The duration of the time under consideration for the stochastic model is the time before any transaction completes its processing and returns its locks, since the start of the process;
2. There is a certain fixed probability of a transaction arriving at the system and requiring at least one item;  
3. The time interval ($\Delta t$), is sufficiently small, such that at most one item is picked up during such an interval; and
4. The total number of objects ($N$), which constitute the database is very large.

Let the probability of an event occurring in the group of objects during a time interval $(t, t + \Delta t)$ be given as follows:

\[ P(\text{an object will be picked up by a transaction}) = \gamma \Delta t + O(\Delta t) \]

where ($\gamma$) is a constant and $O(\Delta t)$ indicates the order of ($\Delta t$), that is ($O(\Delta t)/\Delta t \rightarrow 0$ as $\Delta t \rightarrow 0$). Also,

\[ P(\text{an object will be picked up by a transaction of no-wait group}) = (\alpha \Delta t + O(\Delta t)) P(e) \]

and

\[ P(\text{an object will be picked up by a transaction of wait group}) = \beta \Delta t + O(\Delta t) P(w) \]

where $\alpha = \gamma P(e)$, $\beta = \gamma P(w)$ and, $P(e)$ and $P(w)$ are the probabilities that the transaction picking up an item from the database belongs to the no-wait group and the wait group, respectively.

The Markov assumptions that need to be stated are;

\[ P(\text{more than one item will be picked up by a transaction during} (t, t + \Delta t) = O(\Delta t) \]

\[ P(\text{any of the} n \text{items of the database will be picked up by a transaction during} (t, t + \Delta t) = n \gamma \Delta t + O(\Delta t) \]

At a given moment of time, there are ($N - n$) objects that remain to be picked up. Hence

\[ \lambda_n = (N - n) \alpha \]

Also, of the possible pairs which could be formed, there are $n(N - n)$ which consist of one item in the locked list and the other item in the free list. These are the only pairs which can give rise to the transition ($n \rightarrow n + 1$) and the total probability associated with them is

\[ \lambda_w = n(N - n) \beta \]

Hence,

\[ \lambda_n = \lambda_c + \lambda_w = (N - n)(\alpha + \beta n) \]

where ($n = 0, 1,..., N - 1$).

The probability that during $(t, t + \Delta t)$ the number of objects which have been picked up will remain at $n$ is given by $1 - n \lambda_n \Delta t + O(\Delta t)$.

References