Concurrency Control for Time-constrained Transactions in Distributed Databases Systems

KAM-YIU LAM AND SHEUNG-LUN HUNG

Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon Tong, Hong Kong
Email: cskylam@cityu.edu.hk

The design of concurrency control protocols for time-constrained transactions is complicated due to the requirements to maintain the database consistency and to satisfy the timing constraints of the transactions. In the past few years, various real-time locking protocols have been proposed for different real-time database systems (RTDBS). However, the use of these protocols for distributed real-time database systems (DRTDBS) has received much less attention, even though many RTDBS are distributed in nature. In this paper, two efficient real-time locking protocols are proposed for DRTDBS. The first one, based on dynamic locking, is called Distributed Hybrid Two Phase Locking (DHb2PL). Its performance has been compared in detail with three other distributed real-time locking protocols. The performance results indicate that DHb2PL is much better than the other protocols as a result of a better approach to resolving lock conflicts and its deadlock free property. The second one, called DRT-S2PL, is based on static locking where the locks required by a transaction are assumed to be known before its execution. Its relative performance as compared with DHb2PL is dependent on the proportion of remote locks required by a transaction. DRT-S2PL is more suitable for systems with transactions which have to set a large proportion of remote locks.

Received May 19, 1995; revised September 12, 1995

1. INTRODUCTION

In recent years, there has been a trend to use database technology to support real-time applications as system flexibility and functionality can be greatly enhanced [1–3]. It has become feasible to process complex queries and support the use of complex control systems. In these kinds of applications, some of the transactions are defined with completion time constraints which are usually expressed as deadlines. They are called real-time transactions and the database systems are called real-time database systems (RTDBS). Many RTDBS are distributed in nature. If the database is partitioned and distributed across different sites, the system is called distributed real-time database system (DRTDBS). Some examples of DRTDBS are international financial stock marketing systems, computer integrated manufacturing and battle-field management systems. Research in DRTDBS has received more and more interest in recent years [4–6]. The processing of a real-time transaction in a DRTDBS may require the creation of a number of participants at different sites and the access of data in those sites.

Missing the deadlines in real-time transactions can severely affect their completion values [7]. Based on the impact of missing a deadline, real-time transactions can be classified into three types: hard, firm and soft [2]. Most of the hard real-time transactions are for safety-critical monitoring. Missing deadlines of hard real-time transactions may result in a disaster. Although the impact of missing deadlines of firm real-time transactions will not be catastrophic, completing firm real-time transactions after their deadlines may produce undesirable consequences. Firm real-time transactions should be aborted as soon as possible if their deadlines were missed [8]. The impact of missing the deadline of a soft real-time transaction is less serious but their completion values will decrease dramatically after their deadlines.

One important design issue of RTDBS is transaction scheduling which aims to satisfy the timing constraints of the transactions while maintaining their ACID properties [9]. In conventional real-time systems, the timing constraints of the tasks are ensured by adopting appropriate CPU scheduling algorithms in which different priorities are assigned to the tasks based on their criticality and urgency [10]. ACID properties of transactions can be ensured through concurrency control protocols (CCPs) [9]. Uncontrolled interleaving of transaction execution may violate the database consistency as different transactions may access the same data simultaneously in conflicting modes. CCPs schedule data to transactions and ensure that the database consistency can be restored when transactions are committed. In the last two decades, many CCPs have been proposed. Some of the proposed methods are two phase locking (2PL), optimistic methods (OM) and time-stamp ordering (TO) [11]. It is generally agreed that 2PL gives the best performance for most database applications [9, 11–13].

One major problem in transaction scheduling for RTDBS is the incompatibility of the strategies used in the real-time CPU scheduling algorithms and the CCPs
Concurrency control for time-constrained transactions [2, 14, 15]. None of the well-known CCPs has considered any real-time issues in scheduling the transactions to use the data. The common problems due to this incompatibility are priority inversion [15, 16] and unnecessary restarts. Both of them can severely degrade the system performance.

Concurrency control in DRTDBS is much more complex than in single-site RTDBS [17, 18]. There is still a lack of efficient CCPs for DRTDBS. In DRTDBS, the overhead for concurrency control is high as the data required by a transaction may reside at different sites. Additional issues such as fault tolerance, distributed deadlock, management of replicated data and distributed transactions have to be addressed as well. Unpredictability in transaction response time may result from the requirement to maintain transaction atomicity over participants in different sites and from the management of distributed data. The response time of a transaction is heavily affected by the location of its required data which may not be known before its actual processing. The occurrences of distributed deadlock can dramatically increase the response time of a transaction. The performance of the underlying network also plays an important role on the system performance.

In this paper, the problems in the design of concurrency control protocols for DRTDBS are discussed and new efficient real-time locking protocols are proposed. The performance of the new protocols is examined and compared with three other real-time locking protocols for DRTDBS. The organization of the remaining parts of the paper is as follows. Section 2 reviews some important related work. Section 3 introduces our new hybrid approach and the static locking approach for conflict resolution. Section 4 defines five real-time locking protocols for DRTDBS. Section 5 introduces the model for DRTDBS. Section 6 summarizes the performance results of the protocols. The conclusions of the paper are contained in Section 7.

2. RELATED WORK

Concurrency control in RTDBS has been addressed by a number of recent studies. Different real-time CCPs have been proposed for single-site RTDBS by modifying some well-known non-real-time CCPs to incorporate transaction priority information to resolve data conflicts between transactions [6, 16, 19–25]. However, most of them are restricted to single-site RTDBS [16, 19–21, 23]. Concurrency control in DRTDBS has received comparatively much less attention.

Most of the protocols proposed in the literature are based on 2PL owing to its simplicity and popularity in conventional database systems. In real-time locking protocols, there are three basic methods to resolve lock conflicts between transactions with different priorities. The first method is blocking with priority inheritance (PI) [26] in which the priority of a lower priority transaction will be raised up to the priority of the transaction whose priority is the highest among all transactions which are blocked by this low priority transaction. By raising the priority of the lock holder, it is expected that the blocking times of blocked transactions can be reduced as now the lock holder will not be pre-empted by other intermediate priority transactions from using the CPU. The problem with PI is that altering the priority of a transaction affects the schedulability of the CPU scheduling algorithm. In addition, deadlock is possible in PI.

The second method is to restart lower priority transactions when some of their seized locks are requested by a higher priority transaction, e.g. High Priority Two Phase Locking (H2PL) [14]. If the priorities of the transactions are assumed to be unique, H2PL is deadlock free. The biggest problem with H2PL is the heavy overhead to restart lower priority transactions (in fact some of them are not necessary). Frequent restarts greatly increase the system workload and degrades the system performance especially when the system workload is already heavy [10, 15]. A variant of the restart approach is called Conditional Priority Inheritance (CPI) in which a condition is defined for choosing between restarting transaction and blocking with priority inheritance [26]. One simple way to define the condition is based on the proportion of completion of the lower priority lock holding transaction. If this proportion is greater than a certain pre-specified value, priority inheritance will be used. Otherwise, the lower priority lock holder will be restarted. Similar to PI, deadlock is possible in CPI.

The last method is lock reservation. If the required locks of a transaction are known before the actual processing of its operations, the locks may be reserved first so as to prevent the lower priority transactions from getting them. Some examples of this kind of protocols are Priority Ceiling Protocols (PCPs) [15, 22]. PCPs are suitable for the RTDBS with hard real-time transactions as the blocking times of higher priority transactions are bounded [16].

The study of CCPs in DRTDBS has attracted less attention in the past. Sha [15] suggested the use of a modular approach so that no remote locking is required by assuming that the transactions can be broken into different atomic units. In Ulus [6], the issues of site failure and link failure on the performance of real-time locking protocols have been examined. One weakness of Ulus's work is that the overhead for distributed deadlock detection was not included. Son [5] has adopted the notion of epsilon serializability [27] to reduce the blocking times for committing a distributed transaction. The additional overhead in the management of replicated data has been studied in Ulus [24]. Nearly all previous studies have neglected the use of static locking approach [9] for concurrency control in DRTDBS.

3. NEW APPROACHES TO SOLVING CONFLICTS

3.1. Hybrid approach

Although CPI and PI have been shown to be better than
H2PL for single-site RTDBS [26], they have neglected the impact of blocking transactions on the probabilities of lock conflicts and deadlock. It is commonly agreed that blocking lock-holding transactions may induce more blocking and result in chain of blocking and extensive deadlock in a high data contention environment [28–31]. This will greatly degrade the performance of a RTDBS. Once a blocking chain is formed, more blocking will be induced. Consequently, the probability of deadline missing will be dramatically increased even though under normal conditions it is very low.

One effective way to prevent a chain of blocking is to use cautious waiting in which the length of a blocking chain is limited [32]. In our hybrid approach, the concept of restarting lower priority transactions, priority inheritance and cautious waiting are integrated so as to minimize the overhead to restart and at the same time to prevent a harmful chain of blocking. A chain of blocking is less harmful if the dependencies amongst the transactions are in the same order as their priorities. Thus, in our hybrid approach, whenever a higher priority lock requesting transaction (the lock requester) is detected to have a lock conflict with another lower priority lock holding transaction (the lock holder), the lock requester will not be blocked and one of the conflicting transactions will be restarted in order to break the chain if:

1. The lower priority lock holder is being blocked.
2. The higher priority lock requester is blocking another higher priority transaction.
3. The slack time of the lock requester is smaller than the remaining execution time of the lock holder.

The slack time of a transaction is defined as the deadline of the transaction minus its remaining execution time. Conditions (1) and (2) are used to prevent the formation of a blocking chain with a higher priority transaction being blocked by more than one transaction. If either condition (1) or condition (2) is true, the lowest priority transaction in the chain will be selected for restart to break the chain. Condition (3) is aimed to minimize the amount of workload spent on the deadline missing transactions. If the slack time of the lock requester is smaller than the remaining execution time of the lock holder, blocking the lock requester will cause it to miss its deadline. It is important to the system performance if deadline missing can be detected earlier [33]. If all the defined conditions are false, blocking with priority inheritance will be used to minimize the blocking time of the higher priority transaction.

Since the conditions for transaction restart in our hybrid approach are tighter than H2PL, the number of transaction restarts is smaller. On the other hand, as the use of priority inheritance is restricted, its impact on the overall system performance will not be as serious as that in PI. Comparing with CPI and PI, our hybrid approach is free of deadlock. The only assumption required for our hybrid approach is that the total execution time of a transaction can be estimated precisely. This assumption has been used in many real-time CCPs including CPI. It is valid for most real-time transactions as their behavior and resource requirements are predictable [4, 34].

3.2. Static locking approach

If the required locks of a transaction are known before processing, a static locking approach may be used in which the locks are reserved for higher priority transactions. Although this assumption may not be valid for most transactions in conventional database systems, it is valid for many real-time transactions as their behavior is usually well-defined [4, 15, 34]. For example, in a nuclear power plant management system, some of the transactions are responsible for the updates of the temperatures of the nuclear engines. Their data requirements are predefined. Actually, many previous studies on concurrency control for RTDBS have made a similar assumption or even tighter assumptions. In PCPs, the required locks and the priorities of the transactions have to be known before their arrivals [15].

In the static locking approach, each lock in the database is assigned a priority which equals that of the highest priority transaction among all transactions which are waiting for or using that lock. Before the processing of a transaction, all the locks to be accessed by a transaction have to be set in appropriate modes. If any of its required locks: is in conflicting mode or has a priority higher than that of the requesting transaction none of its required locks will be set and the transaction will be blocked. However, for those locks with lower priorities than that of the requesting transaction, the priorities will be updated. Using this approach, a higher priority transaction will only be blocked by a lower priority transaction which has obtained its locks before the arrival of the higher priority transaction. Since all lock conflicts are resolved by blocking, no transaction has to be restarted due to lock conflict.

4. DISTRIBUTED REAL-TIME LOCKING PROTOCOLS

In this section, five real-time locking protocols will be introduced for concurrency control in DRTDBS. They are based on the hybrid approach, the static locking approach, PI, CPI and H2PL. It is assumed that the lock schedulers are partitioned across different sites with the scheduler at each site being responsible for the management of the locks at that site. It is assumed that each transaction is started with the creation of a process called the coordinator at its site of origin. Other participants are created at different sites where operations need to be processed. The priorities of the coordinator and the participants are the same as their originating transactions.

4.1. Distributed high priority two phase locking (DH2PL)

In a DRTDBS, the overhead and the time required to
restart a transaction can be substantial as it may have participants at different sites. This is especially true if the transaction is near-to-complete. Thus, it is always beneficial to resolve lock conflicts by blocking if the lock holding transaction is committing even if the priority of the lock requesting transaction is higher. A transaction is committing if all its operations have been completed and the atomic commitment protocol is performing [9]. The following defines how lock conflict is resolved in DH2PL:

\[ \text{LockConflict}(t_r, t_h) \]
\[ \text{if } \text{Priority}(t_r) > \text{Priority}(t_h) \]
\[ \text{if } \neg \text{committing}(t_h) \]
\[ \text{if } \text{local}(t_h) \]
\[ \text{local restart of } t_h \]
\[ \text{else } \text{global restart of } t_h \]
\[ \text{endif} \]
\[ \text{else } \text{global restart of } t_h \]
\[ \text{endif} \]
\[ \text{else } \text{block } t_r \]
\[ \text{endif} \]
\[ \text{endif} \]

where committing(t_h) is true if t_h is committing. Local(t_h) is true if t_h has only one participant which is located at its originating site. Local restart means that the participant of the transaction, which is located at its site of origin, has to be restarted. If the transaction has participations in sites other than its site of origination, a global restart is required. All its participants have to be restarted.

4.2. Distributed Priority Inheritance and Distributed Conditional Priority Inheritance (DPI and DCPI)

The greatest problem in extending PI and CPI for DRTDBS is the management of the inherited priorities of the transactions. Since the lower priority transactions may have participants in different sites, the priorities of all the participants have to be updated. A broadcast approach can be used.

\[ \text{LockConflict}(t_r, t_h) \]
\[ \text{if } \text{Priority}(t_r) > \text{Priority}(t_h) \]
\[ \text{if } \text{C}1 \]
\[ \text{block } t_r \]
\[ \text{Broadcast UpdatePriority}(t_h, \text{priority of } t_r) \text{ to all sites} \]
\[ \text{else } \text{if } \text{local}(t_h) \]
\[ \text{local restart of } t_h \]
\[ \text{else } \text{global restart of } t_h \]
\[ \text{endif} \]
\[ \text{endif} \]

\[ \text{UpdatePriority}(t_h, \text{priority}) \]
\[ \text{if } \text{Priority}(t_h) < \text{priority} \]
\[ \text{the priority of } t_h^{s} \text{participants:}=\text{priority} \]
\[ \text{endif} \]

In DPI, C1 is set to be TRUE. In DCPI, C1 is set to be:
\[ (L_{\text{th}} - X_{\text{th}} \leq h) \text{ or committing}(t_h) \]

where \( L_{\text{th}} \) and \( X_{\text{th}} \) are the total number of steps of \( t_h \) and the number of completed steps of \( t_h \) respectively. h is a pre-defined threshold value for choosing between restart and blocking of the lock requesting transaction.

4.3. Distributed Hybrid Two Phase Locking (DHb2PL)

Similar to DH2PL in DHb2PL, if a transaction is committing, it will not be restarted even though a lock conflict is occurred.

\[ \text{LockConflict}(t_r, t_h) \]
\[ \text{if } \text{Priority}(t_r) > \text{Priority}(t_h) \]
\[ \text{if } \text{C}1 \land \neg \text{committing}(t_h) \]
\[ \text{if } \text{local}(t_h) \]
\[ \text{local restart of } t_h \]
\[ \text{else } \text{global restart of } t_h \]
\[ \text{endif} \]
\[ \text{else } \text{block } t_r \]
\[ \text{Broadcast UpdatePriority}(t_h, \text{priority of } t_r) \text{ to all sites} \]
\[ \text{endif} \]
\[ \text{endif} \]

\[ \text{UpdatePriority}(t_h, \text{priority}) \]
\[ \text{if } \text{Priority}(t_h) < \text{priority} \]
\[ \text{the priority of } t_h^{s} \text{participants:}=\text{priority} \]
\[ \text{endif} \]

C1 is TRUE if:
1. \( t_h \) is being blocked; or
2. \( t_r \) is blocking a higher priority transaction; or
3. the slack time of \( t_r \) is smaller than the remaining execution time of \( t_h \).

The implementation problems of DHb2PL are: the lack current blocking information of the transactions at each site in a distributed environment, and no means to accurately estimate the remaining execution time and slack time of a transaction without prior knowledge of the location of its required data. The time to process an operation is heavily dependent on the location of its required data. Obviously, it will be much longer if it has to access remote data. This kind of operation is called a remote operation while those which only need to access local data are called local operations. There are different alternatives for estimating the remaining execution time and the slack time of a transaction by examining the number of remaining remote operations.
1. Minimal Case (DHb2PL-Min): all the remaining operations of the transaction are assumed to be local.
2. Mean Value Case (DHb2PL-MC): the number of remaining remote operations of the transaction is assumed to be equal to the mean number of remote operations in a transaction.
3. Worst Case (DHb2PL-WC): all the remaining operations of the transaction are assumed to be remote.
4. Always Abort (DHb2PL-AA): the lock requesting transaction will be aborted whenever a chain of blocking is going to form with a higher priority transaction being blocked by more than one lower priority transaction.

The current blocking status of a transaction is kept in the originating site of the transaction. If the lock conflict does not occur at the originating sites of the conflicting transactions, a message has to be sent to their originating sites to check their current blocking status.

4.4. Distributed real-time static locking (DRT-S2PL)

Lock checking in the static locking approach has to be performed in a critical section. This is impractical for a distributed environment as it will greatly degrade the system performance. If the checking is done at each site one by one, there is a possibility of distributed deadlock. One solution is to use the sequential locking approach in which each site is defined with a unique order, based on their site identity (site_ID). Checking and setting of locks are started at the site with the smallest site_ID. As the setting of locks in DRT-S2PL at each site is in an all-or-nothing principle, no lock holding transactions will be blocked by other transactions if they have set some locks at that site. All the transactions are blocked in the same order of the site_IDS. It means that a transaction can only be blocked by another transaction, t_i in a site with site_ID of i if t_i has obtained all its locks before sites with site_ID small than i + 1.

\[
\text{LockRequest}(t_i, \{ \text{the locks to be accessed by } t_i \})\{ \\
\text{No\_Conflict} = \text{TRUE} \\
\text{for} \text{ site\_ID} = 1 \text{ to } n \\
\quad \text{for} x_i \in \{ x \mid \text{the locks to be accessed by } t_i \land \text{location of } x_i = \text{site\_ID} \} \\
\quad \quad \text{if } x_i \text{ is locked by another transaction in conflicting mode} \\
\quad \quad \quad \text{No\_Conflict} = \text{FALSE} \\
\quad \quad \quad \text{priority}(x_i) < \text{priority}(t_i) \\
\quad \quad \quad \text{priority}(x_i) := \text{priority}(t_i) \\
\quad \quad \text{endif} \\
\quad \text{endfor} \\
\text{if No\_Conflict = TRUE} \\
\quad \text{set lock on } x_i, \forall x_i \in \{ x \mid \text{the locks to be accessed by } t_i \land \text{location of } x_i = \text{site\_ID} \} \\
\text{endif} \\
\text{endfor}
\]

4.5. Distributed deadlock in DRTDBS

DCPI and DPI are subjected to the problems of local and distributed deadlocks. Although the detection of local deadlock is trivial, solving distributed deadlock is much more difficult. Recent studies indicate that the detection of distributed deadlocks requires heavy workload overheads [35]. Building global dependencies of transactions requires heavy overheads and transmission of large number of messages. Up to now, the most common method used for conventional distributed database systems is the simple time-out method in which a time-out period (TOP) is defined. Periodically, the block queue in each site will be checked. This period is called time-out checking period (TCP). If a transaction has been blocked for a period longer than TOP, it is considered to be involved in a deadlock and will be restarted. The problem in using the time-out method is the difficulty in determining the optimal values of TOP and TCP. If the values of TOP and TCP are not well chosen, the system performance will degrade greatly. This impact is especially serious for DRTDBS due to the importance of timely execution of transactions.

5. MODELLING OF DISTRIBUTED REAL-TIME DATABASE SYSTEMS

5.1. Processing of real-time transactions in DRTDBS

A real-time transaction is defined as a set of operations with constraints on its completion time. The processing of a real-time transaction is dependent on the locking method used. For those protocols based on dynamic locking, e.g. DHb2PL, DH2PL, DPI and DCPI, its execution can be divided into two phases, the locking phase and the commitment phase. The locking phase of a transaction is started with the creation of its coordinator at the site of origination of the transaction. Participants of the transaction are created in those sites where its operations have to be performed and the transaction does not have a participant already. During the locking phase, the operations of a transaction are processed sequentially. It is assumed that the operations of a transaction are totally ordered. After obtaining its required locks in appropriate modes, the processing of an operation will be started. After all the operations have been completed, the transaction enters the commitment phase in which an atomic commitment protocol will be performed.

The execution of a transaction with DRT-S2PL is very different from that based on dynamic locking approach. It is divided into the locking phase, the execution phase and the commitment phase. The locking phase is to set locks. Once all its required locks have been set, the
transaction enters the execution phase in which its operations are processed one by one. The commitment phase is the same as that in the dynamic locking approach.

5.2. Performance model of DRTDBS

The performance model for the DRTDBS consists of four inter-connected sites. There is no global shared memory in the system. Communications between the sites is by passing messages over communication links. Sending messages into and receiving messages from the network are performed by a communication interface at each site. Since, in this we are interested in real-time concurrency control protocols, the issues of supporting real-time communication and the impact of different network issues on system performance will not be addressed. It is assumed that the network has enough capacity to support the transmission of messages. Each message can be delivered in a finite amount of time and is received in the same order as its delivery. In order to eliminate the impact of different network configurations on the system performance, the sites are fully-connected. Each link is modeled as a duplex line. The transmission delay of a message is highly dependent on the communication protocol adopted and the type of computer network. In a recent work, Ulusoy [36] has investigated the performance impact of networking in DRTDBS. In order to simplify the model, each communication link is modeled as a constant delay server with same delay time for the transmission of all kinds of messages between any two sites. It is assumed that the system is reliable so that no site or network failure will occur.

The model for each site is shown in Figure 1. It is an open system with aperiodic real-time transactions.

Each site consists of eight components: the transaction generator, the scheduler, the CPU, the block queue, the ready queue, the communication interface, the disk I/O and the local database. The data in a local database may reside on the disk or in the cache. The transaction generator in each site is responsible for creating real-time transactions, one at a time, with exponentially distributed inter-arrival times. The creation of a transaction at a site is independent of the creation of transactions at other sites. Each created transaction will be assigned a unique transaction ID and deadline. If the required data of a transaction is not located in the cache, a disk request is created to retrieve the data from the disk to the cache for processing. The scheduling for disk access is based on the priorities of the transactions and is pre-emptive.

The processing of an operation involves use of the CPU and access to the data in the database. It is assumed that each operation will access one data and the transactions access their required data in a random manner. That is each data will have the same chance of being accessed by an operation and this probability is independent of the location of the data and the originating site of the transaction. The impact of different locality in data references will be studied in Section 6. For write operations, the original values of the data to be updated will be written into the transaction logs first in order to cater for transaction recovery in the case of system failure.

All transactions wait in the ready queue before being selected by the scheduler for processing. The queuing discipline in the ready queue is earliest deadline first (EDF) with pre-emption as this is the most common CPU scheduling method used in the studies of real-time concurrency control protocols. Whenever the processing of a transaction is completed, the scheduler will select the transaction at the head of the ready queue for using the CPU. When a new transaction joins the ready queue, the scheduler compares the priority of the currently executing transaction with that of the newly arrived transaction. The higher priority transaction will be selected for execution and the lower one will be placed in the ready queue.

The scheduler at each site is also responsible for managing the locks at that site. If a transaction has to be blocked (as defined by the adopted concurrency control protocol), it will be placed in the block queue until the lock request can be satisfied. If a transaction has to set a lock at another site, a remote lock request will be sent to the remote site through the local communication interface. If there does not exist a participant for the transaction at the remote site, a participant for the transaction will be created. After the required lock has been set at the remote site, the processing can begin.

It is assumed that the real-time transactions are firm. Before the processing of an operation, the transaction deadline will be checked. If the deadline has already been missed, the transaction will be aborted immediately as completing a missed deadline firm real-time transaction will be harmful. If the transaction has to be restarted due to lock conflict with another higher priority transaction, its deadline will be checked before it is restarted. If it has already missed the deadline, it will be aborted immediately. Undo operations have to be performed to restore the database state based on the transaction logs. All its seized locks will then be released before it is terminated if it has to be aborted. Otherwise, it is queued in the ready queue for processing from its beginning. After all the operations of a transaction have been processed, the transaction enters its commitment phase.
Although reliability and fault tolerance will not be addressed in this study, an atomic commitment protocol is included in our model so as to make the model more realistic. In real-life database applications, it is common to have a commitment protocol as system failures and different kinds of faults are possible. Including a commitment protocol in the model introduces additional overhead for commitment and may affect the performance of the protocols. The commitment protocol adopted in our model is the two phase commitment (2PC) protocol [37] due to its simplicity and popularity. Permanent update of the data will be performed at the second stage of the commitment phase. All the seized locks of the transaction will be released before it is terminated.

6. PERFORMANCE STUDY OF THE PROTOCOLS

6.1. Deadline assignment

It is assumed that the transactions belong to the same application and are described by the same set of parameters such as deadline constraint, criticalness level, service requirement of the CPU, mean number of operations of a transaction and mean inter-arrival time. One common way in setting the transaction deadline is based on the total expected execution time of the transactions [6, 14, 19, 21] such as:

\[ D_i = A_i + S_i \times E_i \]

\( A_i \) is the arrival time of transaction \( i \) and \( E_i \) is the total expected execution time of the transaction. \( S_i \) is the slack factor which is a random variable uniformly distributed between two slack bounds, upper and lower. These two bounds are the characteristics of an application and its transactions. \( E_i \) is affected by the adopted concurrency control protocols. If the dynamic locking protocol is used, \( E_i \) can be estimated as:

\[ E_i = E_p + E_c \]

where \( E_p \) is the time for the locking phase and \( E_c \) is the time for the commitment phase. \( E_p \) can be estimated as:

\[ E_p = (t_l + tp) \times \text{number of local operations} + \{2 \times (CD + 2 \times T_{msg}) + t_l + tp\} \times \text{number of remote operations} \]

The first term of \( E_p \) is the time for processing the local operations of a transaction and the second term is for remote operations. \( t_l \) is the total CPU time for checking, setting and releasing a lock. \( tp \) is the sum of CPU time and the I/O time for processing an operation. \( CD \) is the communication delay of a message between two sites and \( T_{msg} \) is the CPU time to process a communication message.

The time for the commitment phase is affected by the number of data to be updated and the communication delays for the messages. It can be estimated as:

\[ E_c = \{4 \times (CD + 2 \times T_{msg})\} + \text{max(data_to_be_updated_in_a_site)} \]

Two rounds of communication are required to perform 2PC. Since the update of data at different sites can be done concurrently, the maximum period of time required is dependent on the site with the greatest amount of data to be updated.

<table>
<thead>
<tr>
<th>Parameter Description</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean inter-arrival time (ms)</td>
<td>IAT</td>
<td>2000 (exponential)</td>
</tr>
<tr>
<td>Mean transaction size (operations)</td>
<td>TS</td>
<td>12</td>
</tr>
<tr>
<td>Transaction size modifier</td>
<td>TS_M</td>
<td>0.75 to 1.25 (uniform)</td>
</tr>
<tr>
<td>CPU time to process an operation (ms)</td>
<td>( T_{cpu} )</td>
<td>34 (constant)</td>
</tr>
<tr>
<td>CPU time to set a lock (ms)</td>
<td>( T_{set} )</td>
<td>1 (constant)</td>
</tr>
<tr>
<td>CPU time to release a lock (ms)</td>
<td>( T_{rel} )</td>
<td>1 (constant)</td>
</tr>
<tr>
<td>CPU time to check a lock (ms)</td>
<td>( T_{chk} )</td>
<td>2 (constant)</td>
</tr>
<tr>
<td>CPU time to update a data (ms)</td>
<td>( T_{up} )</td>
<td>6 (constant)</td>
</tr>
<tr>
<td>CPU time to write a commit log (ms)</td>
<td>( T_{log} )</td>
<td>4 (constant)</td>
</tr>
<tr>
<td>Lower slack bound</td>
<td>( S_l )</td>
<td>0.5</td>
</tr>
<tr>
<td>Upper slack bound</td>
<td>( S_u )</td>
<td>1.5</td>
</tr>
<tr>
<td>Communication delay for a message (ms)</td>
<td>( T_{comm} )</td>
<td>50 (constant)</td>
</tr>
<tr>
<td>Number of sites</td>
<td>( N_{site} )</td>
<td>4</td>
</tr>
<tr>
<td>Database size (data objects)</td>
<td>DS</td>
<td>800</td>
</tr>
<tr>
<td>CPU time to process a communication message</td>
<td>( T_{msg} )</td>
<td>4 (constant)</td>
</tr>
<tr>
<td>Time-out period (ms)</td>
<td>TOP</td>
<td>1500</td>
</tr>
<tr>
<td>Time-out checking period (ms)</td>
<td>TCP</td>
<td>1500</td>
</tr>
<tr>
<td>CPU time to check for time-out/tran (ms)</td>
<td>( T_{check} )</td>
<td>2</td>
</tr>
<tr>
<td>Local operation probability</td>
<td>( P_{loc} )</td>
<td>0.25</td>
</tr>
<tr>
<td>Cache data probability</td>
<td>( P_{cad} )</td>
<td>0.25</td>
</tr>
<tr>
<td>Number of CPUs in each site</td>
<td>( N_{CPU} )</td>
<td>1</td>
</tr>
<tr>
<td>Data allocation in each site (data ID)</td>
<td>LDB</td>
<td>site 1: 1 to 200, site 2: 201 to 400, site 3: 401 to 600, site 4: 601 to 800</td>
</tr>
</tbody>
</table>
6.2. Performance parameters and measures

The set of model parameters and their chosen basic values are depicted in Table 1.

As we aim at making a general performance comparison, rather than investigating the performance of the protocols under a specific DRTDBS, the values of the parameters should be chosen to allow ease of comparison and the cost of running the simulation model should be affordable. To illustrate the performance differences amongst the protocols, the probability of lock conflict should be high. High lock conflict can be obtained by using a small database and relatively long transactions. The CPU time required to process an operation is application dependent. In our study, these values are based on the values used by others [6, 8, 14, 21]. All these studies use a constant delay in an order of 10th ms for the CPU time to process an operation. The service times to set, check and release a lock have been neglected in many previous studies [6, 8, 14, 21, 24]. However, in many of the studies on concurrency control with 2PL for conventional database systems, it has been found that although the overhead for lock management is small, it can have significant effects on the system performance especially in a high data contention environment [12]. Thus, in our experiments, they are set to be 1ms. In the time-out method, the overhead for deadlock detection is dependent on the number of transactions in the block queue. The CPU time required for checking the blocking time of a transaction should be small and is comparable to that for checking and setting a lock. The communication delay is highly dependent on the types of network, the message size and the distance between each site. In our experiments, it is set to be a constant of 50 ms which is in the order of the common communication delay for most LAN and MAN.

The most important performance measure used is the missing ratio, MR, which is defined as:

\[
MR = \frac{\text{number of transactions with missed deadlines}}{\text{number of transactions processed}}
\]

6.3. Results analysis and discussion

In this section, the performance of the real-time locking protocols introduced in Section 4 are reported. The performance studies are divided into four parts. The first part compares the performance of DCPI and DPI. The second part investigates which method for the estimation of the transaction execution time is the most suitable for DHb2PL. The third part compares the performance of DHb2PL with DCPI and DH2PL. The last part compares DRT-S2PL with HDh2PL which has been found to have the best performance amongst the tested dynamic real-time locking protocols. The simulation time for each experiment is the time to process 5500 transactions with the statistics collected from the first 500 processed transactions being discarded in order to remove the initial bias.
restarting the lower priority transactions. As depicted in Figure 3, under heavy workload, the performance of DCPI is always better than DH2PL no matter which value of $h$ is used as the impact of transaction restarts on the system performance is high. However, in Figure 4 (low workload), the performance of DCPI is only better than DH2PL within the range of $h$ from 0.5 to 0.45. It is because under low workload, more transaction restarts can be tolerated.

In the following experiments, the value of $h$ to be used is 0.33 as it is the optimal value for medium system workload (as found in the above experiments).

6.3.1.2. The impact of time-out period (TOP) and time-out checking period (TCP). Since DPI and DCPI are subjected to the problem of distributed deadlock, two essential parameters affecting their performance are the values of TOP and TCP. In both Figures 5 and 6 it can be seen that DPI is more sensitive than DCPI to changes in TOP and TCP. It is because the probability of deadlock is much higher in DPI than in DCPI. In Figure 5, the optimal value of TCP for DCPI is found to be 1500 ms. Too small a value incurs heavy workload on the system while a large value may cause large amount of deadlocked transactions to become blocked unnecessarily. In the following experiments, the values of TOP and TCP for DCPI are both set to 1500 ms.

6.3.2. Choosing the variants of DHb2PL

Figure 7 depicts the results of the four variants of DHb2PL as defined in Section 4.3. It can be seen that the methods adopted in these variants have significant impact on the performance of DHb2PL. Amongst the variants, DHb2PL-WC gives the best performance while the performance of DHb2PL-AA is the worst, especially under heavy workload. The poor performance of DHb2PL-AA is mainly due to unnecessary transaction aborts. The use of minimal estimation (DHb2PL-Min) or mean estimation (DHb2PL-MC) is too optimistic. In a DRTDBS, the execution time of a transaction is usually much greater than its required total service time especially under heavy workload. Thus, a pessimistic approach (DHb2PL-WC) can give a more accurate estimation provided that the communication delays are not too large.

6.3.3. Performance comparison of DHb2PL, DCPI and DH2PL

As shown in Section 6.3.1, DCPI always performed...
better than DPI, in the following experiments, only DCPI and DH2PL will be compared with DHb2PL. The variant chosen for DHb2PL is DHb2PL-WC as its performance is the best amongst the four variants as shown in the last sub-section.

6.3.3.1. Mean inter-arrival time. One deterministic factor on the system performance is the system workload which can be changed by varying the mean inter-arrival time of the transactions. Figure 8 depicts the relative performance of the three distributed real-time locking protocols (DHb2PL, DCPI and DH2PL) as a function of mean inter-arrival time. Under low workload (large mean inter-arrival time), the performance of DHb2PL, DH2PL and DCPI are similar, with DHb2PL having a slightly better performance. The differences in their performance become more significant under heavier workload. The better performance of DHb2PL as compared with DCPI suggests that distributed deadlock and chain of blocking can have significant impact on system performance. The values of MR for DH2PL increase sharply when IAT is smaller than 2.5 s, indicating that the impact of transaction restarts is more significant when the system workload has reached a high level.

6.3.3.2. Database size and transaction size. In Figure 9, the impact of database size on relative performance is depicted. The performance of DHb2PL is still the best. The values of MR for all the protocols decrease with an increase in database size. Increasing the database size reduces the probability of lock conflict and also the probability of priority inversion.

Reducing the mean transaction size reduces the system workload and the lock conflict probability for a fixed mean inter-arrival time. Figure 10 depicts the result for the system with the mean transaction size reduced to six operations. The values of MR for the protocols are much smaller than those in Figure 8 and the differences in their performance is also smaller. Unlike Figure 8, the performance of DCPI is slightly better than DHb2PL under heavy workload (e.g. IAT < 1 s). It can be explained by the fact that if the transactions are small, the impact of blocking will be less serious. The probability of distributed deadlock and the impact of priority inheritance on the system performance is smaller. On the other hand, using transaction restarts...
to resolve the conflict becomes less preferable as the deadline constraint of the transactions is tighter.

To further investigate the impact of transaction size on their performance, the transaction size is varied while the mean inter-arrival time keeps the same. The result is depicted in Figure 11. Consistent with the results in Figure 10, DHb2PL is better for larger transactions while DCPI performs better for smaller transactions. The sudden rises in the values of MR for mean transaction size greater than 10 for DH2PL is mainly due to heavy system workload which is the result of restarting large transactions.

6.3.3. Probability of local locks. In DRTDBS, it is common to have transactions with a higher probability to access the data in on originating site. In Figure 12, the probability of its requesting locks in its originating site is set to be higher than those in other sites. It is set to be 0.4 and the probability to set lock in each of the other site is 0.2. It can seen that the relative performance of the protocols is also similar to that in Figure 8. The performance of DHb2PL is still the best. The differences in performance between DCPI and DHb2PL are smaller under heavy workload as compared with Figure 8.

Increasing the probability of local locks reduces the overhead in restarting a transaction. Thus, the probability of missing deadline is smaller. Figure 13 depicts the relative performance of the protocols as a function of probability of local locks. The performance of DHb2PL is still the best for the whole range of probabilities except for the probability of about 0.55.

6.3.4. Comparing DRT-S2PL with DHb2PL

Figure 14 compares the values of MR for DRT-S2PL and DHb2PL under different workloads by changing the mean inter-arrival time for a system with local transactions only. DHb2PL outperforms DRT-S2PL for the whole range of workloads. The poor performance of DRT-S2PL is mainly due to higher lock conflict probability as the lock holding time is longer. The impact of transaction restarts becomes less significant as all the transactions can be processed in their originating sites.

Figure 15 depicts the values of MR for the system with global transactions only. Unlike the result in Figure 14, the performance of DRT-S2PL is much better than DHb2PL. Although the probability of lock conflict is also higher with DRT-S2PL, its performance is better.
mining the optimal value for h. The performance of DCPI is usually better than DPI. The relative performance of DCPI as compared with DH2PL is heavily dependent on the existing system workload and the transaction sizes. Under heavy workload conditions, DCPI performs better than DH2PL owing to the blocking nature and the greater impact on system performance due to transaction restarts in DH2PL.

The use of static locking approach for DRTDBS has been neglected in the past. Owing to the more well-defined data requirement of the real-time transactions, it may be more preferable to use real-time static locking for DRTDBS. With the use of lock reservation, no transactions have to be restarted and the blocking time of higher priority transactions can be much predictable. It has been found that the relative performance of DRT-S2PL as compared with the dynamic-based real-time locking protocols is highly dependent on the proportion of remote locks of a transaction. It is suitable for systems where transactions have to make a large proportion of remote locks.

REFERENCES


