

# CONCURRENCY CONTROL IN MOBILE DISTRIBUTED REAL-TIME DATABASE SYSTEMS<sup>†</sup>

Kam-yiu Lam<sup>1</sup>, Tei-Wei Kuo<sup>2</sup>, Wai-Hung Tsang<sup>1</sup> and Gary C.K Law<sup>1</sup>

Department of Computer Science City University of Hong Kong 83 Tat Chee Avenue, Kowloon HONG KONG<sup>1</sup>  
Department of Computer Science and Information Engineering National Chung Cheng University Chiayi, 621 Taiwan, ROC<sup>2</sup>  
(Received December 1998; in final revised form February 2000)

**Abstract**  $\frac{3}{4}$  *With the rapid advances in mobile computing technology, there is an increasing demand for processing real-time transactions in a mobile environment. This paper studies concurrency control problem in mobile distributed real-time database systems (MDRTDBS). Based on the High Priority Two Phase Locking (HP-2PL) scheme, we propose a distributed real-time locking protocol, called Distributed High Priority Two Phase Locking (DHP-2PL), for MDRTDBS. In the protocol, the characteristics of a mobile computing system are considered in resolving lock conflicts. Two strategies are proposed to further improve the system performance and to reduce the impact of mobile network on the performance of the DHP-2PL: (1) A transaction shipping approach is proposed to process transactions in a mobile environment by exploring the well-defined behavior of real-time transactions. (2) We explore the application semantics of real-time database applications by adopting the notion of similarity in concurrency control to further reduce the number of transaction restarts due to priority inversion, which could be very costly in a mobile network. A detailed simulation model of a MDRTDBS has been developed, and a series of simulation experiments have been conducted to evaluate the performance of the proposed approaches and the effectiveness of using similarity for concurrency control in MDRTDBS.*

**Keywords:** Distributed Real-time Databases, Mobile Real-time Databases, Concurrency Control, Data Similarity, Transaction Scheduling

## 1. INTRODUCTION

Recent advances in wireless communication technology have made mobile information services a reality [9, 10, 25]. A number of novel mobile computing systems, such as tele-medicine systems, real-time traffic information and navigation systems, and mobile Internet stock trading systems, are emerging as mobile users require instant access to information using their palmtops, personal digital assistant (PDA) and notebook computers. Mobile computing technology not just only improves the distribution and flow of information, but at the same time, it also greatly increases the functionality of real-time database applications. The realization of “instant” information access over a mobile network relies on real-time processing of transactions and it makes the timeliness of data accesses an important issue. As a result, research on processing soft real-time transactions in *mobile distributed real-time database systems* (MDRTDBS) is receiving growing attention in recent years [2, 14, 21, 32, 35].

Owing to the intrinsic limitations of mobile computing systems, such as limited bandwidth and frequent disconnection, the design of an efficient and cost-effective MDRTDBS requires techniques that are quite different from that in *distributed real-time database systems* (DRTDBS) which are supported with wired networks [18]. It is much more difficult to meet transaction deadlines in a mobile environment as there exist various factors, such as network performance, concurrency control and transaction scheduling, which can seriously affect the transaction performance. Two of the most important performance objectives are how to meet the urgency of transactions and how to satisfy the temporal constraints of database, where temporal constraints refer to the freshness of data objects in the database [27]. Many real-time database applications are used to monitor the status of the objects in the external environment and they must generate timely responses to critical events. For example, in a stock trading system, a late response to a stock analysis transaction may result in a loss of a good trading opportunity. The consequence of missing a transaction deadline in a tele-medicine system for ambulance services may result in a loss of a human life.

One of the most important issues to ensure timeliness of transaction execution is concurrency control. However, the concurrency control protocols for conventional database systems are not suitable to real-time database systems. Real-time transactions are critical and have to be scheduled to meet their deadlines. Conventional concurrency control protocols, such as two phase locking (2PL) and optimistic concurrency control method [6], often schedule transactions on an equal basis. Higher-priority transactions may suffer from an unlimited amount of priority inversion time, where priority inversion is a situation in which a higher-priority

---

<sup>†</sup> Recommended by Patrick O’Neil, Area Editor.

transaction is blocked by a lower-priority transaction [28]. In the past decade, researchers have proposed various real-time concurrency control protocols, e.g., [1, 8, 24, 29, 28, 30, 36], for single-site as well as distributed RTDBS. In particular, [8, 28] proposed the idea of priority inheritance, which lets a lower-priority transaction inherit the priority of a higher-priority transaction which is blocked by the lower-priority transaction, to reduce the number of priority inversions of the higher-priority transaction.

The priority inversion problem may also be resolved by transaction restart. The High Priority Two Phase Locking (HP-2PL) protocol [1] restarts a lower-priority transaction if a higher-priority transaction wants to set a lock which is held by the lower-priority transaction. The priority inversion problem, in general, is also resolved by transaction restart in the optimistic concurrency control protocols. In the optimistic concurrency control with wait 50 (OCC-wait 50) [8], the execution of a transaction is divided into three phases: the read phase, the execution phase and the validation phase. Data conflicts amongst different transactions will be resolved when one of them enters the validation phase. If the number of conflicting transactions, which priorities are higher than the validating transaction, is not greater than 50% of the total number of conflicting transactions, all the conflicting transactions will be restarted and the validating transaction is allowed to enter the write phase and then commit. Otherwise, the validating transaction will be blocked.

Although many researchers, e.g., [1, 8, 13, 24, 28, 36], have done excellent research in concurrency control for single-site and distributed RTDBS, there is little work in concurrency control for MDRTDBS which is a fast growing and important area. In a MDRTDBS, the mobile network imposes a serious time burden on the performance of a MDRTDBS and it also can seriously affect the performance of the adopted concurrency control protocol. Although the concurrency control protocols proposed for DRTDBS can be extended for MDRTDBS, their performance may be very different from that in a DRTDBS [31], due to the unique characteristics of mobile network.

Compared to wired networks, mobile networks are much slow, unreliable, and unpredictable. The mobility of clients affects the distribution of workload in the network and the system. Disconnection between mobile clients and base stations is common [9]. It can seriously affect the probability of data conflicts and the deadline missing probability. The poor quality of services provided by a mobile network can also seriously increase the overheads and affect the effectiveness of a concurrency control protocol in resolving data conflicts [32] as the transactions now require a longer time for completion.

In this paper, we study concurrency control for MDRTDBS. Based on the High Priority Two Phase Locking (HP-2PL) scheme [1], the Distributed HP-2PL (DHP-2PL) protocol is proposed for MDRTDBS. We consider transactions, which are simple flat transactions with read and write operations. This assumption on transaction structure is reasonable for many mobile soft (and firm) real-time applications with web-oriented interfaces, especially for applications running on mobile palmtop computers such as stock monitor systems and traffic information systems. The simplification of transaction behavior and characteristics may result in good strategies in designing real-time transaction scheduling algorithms to improve the system performance. Two new strategies are proposed to further improve the performance of the DHP-2PL. We first propose the transaction shipping approach to reduce the performance dependency of a concurrency control protocol on the performance of the underlying mobile network. We then adopt the notion of similarity to resolve data conflicts, which can be very costly in a mobile environment. Different issues in the design of the similarity-based concurrency control protocol are suggested. A detailed model of a MDRTDBS has been developed, and a series of simulation experiments is conducted to demonstrate the capability of the proposed approaches, for which we have obtained very encouraging results.

The main contributions of the paper are: (1) To our best knowledge, this is one of the first papers on concurrency control for MDRTDBS with a detail performance study. (2) We have designed a distributed real-time locking protocol in which the characteristics of a MDRTDBS are considered for resolving lock conflicts. (3) A transaction shipping approach is proposed to reduce the impact of mobile network on the system performance and the performance of the real-time locking protocol. Different issues in the implementation of the transaction shipping approach are discussed in details. (4) We have extended our real-time locking protocol to be a similarity-based protocol. Different issues in the design of the real-time similarity-based protocols are suggested. (5) A detailed simulation model is developed, in which the mechanisms in mobile communications such as mobility of clients and disconnection, and real-time transaction processing, are included. (6) Simulation experiments have been performed to investigate the performance of the proposed protocols and approaches, and the effectiveness of using similarity as the correctness criterion for concurrency control in MDRTDBS.

The rest of this paper is organized as follows: In Section 2, a model of MDRTDBS is defined. In Section 3, the concurrency control protocol for MDRTDBS is discussed, and the DHP-2PL is introduced. In Section 4, the transaction shipping approach for MDRTDBS is introduced. In Section 5, the notion of similarity is applied for concurrency control in MDRTDBS, and a similarity-based real-time locking protocol is proposed based on the DHP-2PL. In Section 6, the performance MDRTDBS model is described. The baseline setting, the parameters, and the performance measures are given. The simulation results are presented with discussions. Finally, the conclusions and future research directions are given in Section 7.

## 2. MODEL OF A MOBILE DISTRIBUTED REAL-TIME DATABASE SYSTEM

### 2.1. System Architecture

A MDRTDBS consists of four major components: the mobile clients (MCs), the base stations, the mobile network, and the main terminal switching office (MTSO) [16, 22], as shown in Figure 1. The mobile network is assumed to be a radio cellular network and the entire service area is divided into a number of connected cell sites. Within each cell site, there is a base station, which is augmented with a wireless interface to communicate with the MCs within its cell site. The cellular radio network is assumed to be the Global Systems for Mobile Communication (GSM) 900<sup>1</sup> in which two sub-bands of 25 MHz each are defined. One of the sub-bands is 890 to 915 MHz and is for uplink (for the mobile clients to transmit signals to the base station). Another sub-band is 935 to 960 MHz and is for downlink. Within each sub-band, a number of channels are defined for transmitting radio signals which can be data or control signals. Each channel is divided into several time frames by using the time division multiple access (TDMA) method. Usually, the data transmission rate of a channel is between 9.6 to 14.4 Kbps.<sup>2</sup>

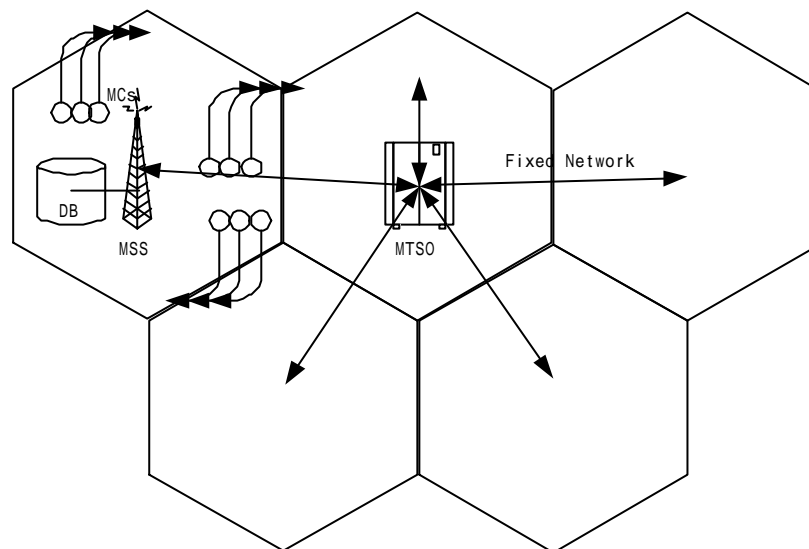


Fig. 1: System Architecture of the Mobile Distributed Real-time Database System

The base stations at different cell sites are connected to the MTSO by a point-to-point wired network. Thus, the communications between the base stations and the MTSO are much more efficient and reliable than the communications between the base stations and the mobile clients. The MTSO is responsible for active call information maintenance, performance of handoff procedure, channel allocation, and message routing. Attached to each base station is a real-time database system containing a local database which may be accessed by transactions issued by the MCs within the cell site or from other cell sites via the MTSO.

An MC may move around within the same cell site or cross the cell border into another cell site. Periodically, it sends a location signal to its base station through an uplink channel. The strength of signal received by a base station depends on several factors, such as the distance between the MC and the base station, and the surrounding buildings. When an MC is crossing the cell border, the strength of signal received by a base station will become very weak. If the strength of the signal is lower than a certain threshold level, the MTSO will be notified, and then the MTSO will perform a handoff procedure. It sends out requests to all the base stations, and the base stations respond by returning the strength of the location signals received from the MC. The MTSO will then assign the MC to the base station, which has received the strongest signal. Usually, this is the base station, which is responsible for the cell site where the MC is entering.

An MC transaction may need to access databases located at several base stations. The MC first issues a call request to the base station of its current cell site. A channel is granted to the MC after the completion of a set-up procedure. The execution of the setup procedure incurs a fixed overhead as it involves communications

<sup>1</sup> In this paper, we focus our discussions on concurrency control of data access over a radio cellular network although there exist several other mobile communication technologies such as satellite and wireless ATM. They are much more expensive.

<sup>2</sup> Although the cellular radio network technologies are improving, and different new standards such as GSM 1800 and DCS 1800, which can support more channels, are emerging, the bandwidth is still a great concern in a cellular radio network.

amongst the MC, the base station, and the MTSO. Since the number of channels between a base station and its MCs is limited, it is possible that the channel request may be refused due to unavailability of free channels. The queuing for channels is according to the priorities of the transactions. If the number of attempts exceeds a specific maximum number, the channel request and the requesting transaction will be aborted. Due to channel contention and slow (and unreliable) communication, the time required to establish a channel is unpredictable. Once a channel has been established, the transaction will be sent out through the RF transmitter from the MC to the base station. When an MC is crossing the cell site border while it is communicating with its base station, a new channel will be created by the MTSO with its newly assigned base station after a setup procedure is completed. However, it is possible that there is no free channel available at the new cell site. In this case, it will retry for a number of times. If it is still not able to get a channel, the transaction will be aborted. Due to noise and interference, the signal, which carries data, may be corrupted while it is being transmitted. In this case, the data will be re-transmitted. If the transmission of signals is corrupted consistently after several times, a disconnection may have occurred. For that case, the transaction may need to wait until a new channel is granted before it can proceed. Because of high error rate and non-stability of signal transmission, the effective data transmission rate is unpredictable.

## 2.2. Database and Transaction Models

The entire database is partitioned into local databases and distributed at different base stations. The databases consist of two types of data objects: *temporal* and *non-temporal*. Temporal data objects are used to record the status of the objects in the external environment. Each temporal data object is associated with a timestamp, which denotes the age of the data object. If a transaction may update a temporal data object, then the transaction is given a timestamp when it is initiated. If the transaction commits successfully before its deadline, the timestamp of any data object, which is updated by the transaction, will be set as the timestamp of the transaction.

The validity of a temporal data object is defined by an *absolute validity interval (avi)* [27, 36]. A temporal data object satisfies the *avi* constraint if its age is up to date, i.e., the difference of the current time and the age is no more than the *avi*. A *relative validity interval (rvi)* may be given to a transaction which requires that the maximum age difference of the data objects read by the transaction is not larger than *rvi* [27, 36]. Non-temporal data objects are either derived by operations of transactions or are statically set during system initialization.

Transactions from the MCs are assumed to be simple flat transactions with a collection of read and write operations [6]. In between the operations of a transaction, control statements may be defined to control the logic flow of the transaction. Examples of such application systems are jockey-club betting systems, Internet programmed stock trading systems, traffic navigation and information systems, etc.

Each transaction is given a deadline and a criticality. The priority of a transaction is derived based on its deadline and criticality. It is assumed that the EDF algorithm is used for scheduling the transactions in using the CPU. It is assumed that the transactions are firm real-time [34]. If the system cannot complete a transaction before its deadline, the transaction will be aborted. Operations may access data objects residing at different base stations. Thus, a transaction may have several processes, called *transaction processes*, at different base stations for its execution. When an operation of a transaction accesses a data object residing at another base station, the operation will be routed to the base station via the MTSO and a new transaction process will be created if there is no process at that site for the transaction. When all the operations of a transaction have been processed, a commit protocol will be performed to ensure the failure atomicity of the transaction processes of the transaction. It is assumed that the well-known two phase commit protocol is adopted because of its simplicity and well-known performance characteristics<sup>3</sup> [6].

## 3. CONCURRENCY CONTROL PROTOCOL FOR MDRTDBS

In the design of concurrency control protocols for MDRTDBS, there are two important considerations: (1) how to minimize the cost and overheads for resolving data conflicts; and (2) how to minimize the impact of mobile network on the performance of the protocol. Restarting a transaction is highly expensive in a mobile environment. Although priority inheritance [8] is effective in managing the priority inversion problem in single-site RTDBS, it may not be effective in a MDRTDBS due to the slow network. It may take a long time before the priority of a transaction is “inherited”. Furthermore, deadlock is possible when priority inheritance is used. Deadlock is highly undesirable to real-time systems, especially in a distributed environment. It is not only because it greatly increases the response time of (deadlocked and other affected) transactions, it also wastes a lot

---

<sup>3</sup> Other real-time commit protocol may be used [Gupt96, Gupta97]. However, in here, we assume a non-real-time commit protocol in order to simplify the analysis.

of system resources. The detection and resolution of a deadlock in a distributed environment may also consume a lot of resources. The most common method for distributed deadlock resolution is time-out, which is obviously not suitable to RTDBS due to the difficulty in determining the appropriate timeout period.

Although the optimistic concurrency control protocols have been shown to give a good performance in single-site RTDBS [8, 20], they may not be suitable to MDRTDBS. The validation test required in the optimistic concurrency control protocols can be very complex in a distributed environment [17], and it will be more complicated in a mobile network.

For the purpose of this paper, we choose to adopt a lock-based approach in which both transaction restart and priority inheritance are used to resolve the problem of priority inversion. By extending the well-known HP-2PL [1], a distributed extension of HP-2PL, called *Distributed HP-2PL (DHP-2PL)*, is proposed for MDRTDBS. In the protocol, special considerations are paid on the characteristics of the mobile network, e.g. low bandwidth and frequent disconnection. For example, special attention is given on how to reduce the number of transaction restarts, which can be very costly due to the low bandwidth. If a transaction is committing, it will not be restarted even if it has a lock conflict with a higher-priority transaction. Instead, priority inheritance is used to reduce the blocking time of the higher-priority transaction. Priority inheritance is applied to the transaction process of the lock-holding transaction at the conflicting site. So, the time required to raise up the priority of a transaction process can be very short. Unlike many other lock-based protocols, such as those simply adopt priority inheritance, DHP-2PL is free of deadlock as priority inheritance is only restricted for resolving lock conflict with committing transactions.

As we can see in the model description in Section 2, the system consists of both a mobile network (connecting the mobile clients and the base stations) and a reliable wired network (connecting the base stations and MTSO). Since mobile network is vulnerable to disconnection, strategies for resolving data conflict should consider the network quality connecting the conflicting transactions. A different strategy should be used if a mobile network instead of a reliable wired network connects the conflicting transactions especially when it is suspected that disconnection may have occurred. In the DHP-2PL, a *cautious waiting* scheme is included to resolve this kind of lock conflicts.

DHP-2PL is a distributed locking protocol. The local database system at each base station has a lock scheduler, which manages the lock requests for the data objects residing at the base station. The definition of the DHP-2PL is as follows, where  $T_r$  and  $T_h$  are the lock-requesting transaction and the lock-holding transaction, respectively:

```

Lock Conflict ( $T_r, T_h$ )
Begin
  If      Priority( $T_r$ ) > Priority( $T_h$ )
    If       $T_h$  is not committing
      If       $T_h$  is a local transaction
        Restart  $T_h$  locally
      Else
        Restart  $T_h$  globally
    Endif
  Else
    Block  $T_r$  until  $T_h$  releases the lock
    Priority( $T_h$ ) := Priority( $T_r$ ) + fixed priority level
  Endif
Else
  Block  $T_r$  until  $T_h$  releases the lock
Endif
End

```

A transaction is local if it only accesses data objects resided at one base station. Otherwise, it is a global transaction. Similar to the HP-2PL, the DHP-2PL uses a transaction restart mechanism to resolve lock conflicts between non-committing transactions. Restarting a local transaction is simply done by restarting the transaction process at the conflicting base station. To restart a global transaction, restart messages are sent to the base stations where some operations of the global transactions are executing or have executed. Global restart takes a much longer time and requires much higher overheads. Thus, the number of transaction restarts, especially a global one, should be minimized. A reasonable approach is to allow a committing transaction to hold a lock until it has finished the commit procedure even though a higher-priority transaction is requesting the lock. Although this approach may create the priority inversion problem, the blocking time of the higher-priority transaction will not be long if the committing transaction is assigned a sufficiently high priority by using priority inheritance. The priority of the committing transaction will be raised up by two factors. Firstly, its priority will be at least as high as the highest priority of all of its blocked transactions, and, secondly, a fixed priority level should be added to its priority to make it even higher than all other executing transactions. The purpose is to

finish the committing transaction as soon as possible. The time required to raise up the priority of a transaction process should be very short as both conflicting processes are located at the same site. No deadlock is possible for the priority raising of any committing transaction. It is because the committing transaction will not be blocked by any other executing transaction as it will not make any lock request during its commitment.

A common characteristic of mobile networks is that disconnection between a mobile client and its base station is common. In processing a transaction, the control of a transaction may flow between its processes at the base stations and the process at its originating mobile client. In case a disconnection occurs while a transaction is locating at the mobile client (the control flow of the transaction is at the mobile client), the impact of the disconnection on the system performance can be very serious due to chain of blocking. It does not only greatly increase the deadline missing probability of the disconnected transaction, other transactions, which are directly or transitively blocked by the disconnected transaction, will also be affected. The result may be *fruitless blocking* which is a situation where a blocked transaction is finally aborted due to deadline missing. In the above protocol, the problem of fruitless blocking may occur when a lower-priority transaction is blocked by a higher-priority transaction which is a disconnected transaction. To minimize the impact of disconnection and the probability of fruitless blocking due to disconnection, we may use a *cautious waiting* scheme in which a higher-priority transaction is restarted by a lower-priority transaction due to data conflict if the higher-priority transaction is suspected to be a disconnected transaction.

Let each executing transaction process be associated with a location indicator. When a transaction is at the mobile client or it is waiting to move back to the mobile client, the location indicator of its processes will be set as "mobile client". Otherwise, it is set as "base station". When the location indicator of a transaction process is "mobile client", the transaction is vulnerable to disconnection. The following summarizes how the cautious waiting scheme is incorporated into the DHP-2PL:

```

If      (the priority of the lock-requester > the priority of the lock-holder ) and
        (the lock-holder is not committing)
        Restart the lock-holder (globally or locally, depending on the type of the transaction)
Else
    If      location indicator of the lock-holder is "mobile client"
        If      the time already spent at the client side > threshold
                Ping the mobile client where the lock-holder is residing
                /* the base station sends a message to the mobile client
                to test whether the mobile client is disconnected or not */
                If      no response from the mobile client
                        Restart lock-holder
                Else
                        Block the lock-requester
                /* repeat the checking after another threshold */
                Endif
        Else
                Block the lock-requester
                /* the checking will be performed again when the time already spent at the
                client side is greater than the threshold value */
        Endif
    Else
        Block the lock-requester.
    Endif
Endif

```

The *threshold* is a tuning parameter. It is a function of the average performance of the mobile network under normal situations. If a transaction has been at the mobile client for a long time, e.g., greater than the threshold value, and its base station cannot communicate with the mobile client currently, disconnection is assumed. The lock-holding transaction will be restarted even though its priority is higher than the priority of the lock-requesting transaction. Although restarting the lock-holding transaction will make it have a high probability of missing deadline, the restart will not affect the system performance significantly as the lock-holding transaction is likely to miss its deadline due to disconnection. On the other hand, restarting the lock-holding transaction can increase the chance of meeting the deadline of the lock-requesting transaction. Otherwise, it is highly possible that both of them will miss their deadlines. Note that simply making the assumption of disconnection based on pinging the mobile client may not be sufficient as mobile networks are subjected to different transient communication failures, which are much less harmful than disconnection. A transient communication failure will only last for a very short time and can usually be solved by data retransmission.

## 4. STRATEGIES FOR PROCESSING TRANSACTIONS IN MDRTDBS

In this section, we discuss different strategies for processing transactions in a MDRTDBS. A new approach, called *transaction shipping*, is proposed. The goal of the transaction shipping approach is to reduce the impact of mobile network on the performance of the DHP-2PL and to improve the system performance.

### 4.1. Data Shipping and Query Shipping

There are two well-known approaches for processing transactions in a client-server database system: query shipping and data shipping [5]. In the data shipping approach, a transaction initiated by a client will be processed at the client. While the transaction is processing, the client sends data requests, which are required by the transaction, to the database server. The server responds to the requests by sending the required data objects to the client. The processing of the transaction will be completed at the client.

In the query shipping approach, the client sends queries to the database server for the transaction, instead of data requests. Once the server receives a query, it processes the query and sends the results back to the client. In the query shipping approach, the communication cost and the buffer space required at the client side are smaller than that in the data shipping approach. Also, the query shipping approach provides a relatively easy migration path from an existing single-site system to the client-server environment since the database engine can have a process structure similar to that of a single-site database system. On the other hand, the data shipping approach can off-load functionality from the server to the clients. This may improve the scalability of the system and balance the workload in the system. The responsiveness of the system can also be improved by caching data at the client. For that case, transactions may be processed locally at the client if they can find their required data.

Although the query and data shipping approaches are suitable to client-server database systems connected with reliable high-speed networks, they may not be suitable to MDRTDBS, which run over a mobile network with a bandwidth rated in the range of 9.6kbps to 14.4kbps. The communication overheads of these two approaches are often high because processing of each operation may require a communication (and the establishment of a new communication channel) between the mobile client and the database server. Remember that communication channels in a MDRTDBS are always limited resources. Furthermore, the data shipping approach usually requires the transmission of a large volume of data objects and the management of data objects at the client caches. Since the mobile network is likely to be the bottleneck resource, the transmission delay can be very long and the resulting effect will be a higher probability to miss deadlines.

### 4.2. Transaction Shipping Approach

A better transaction processing strategy for MDRTDBS should be designed to reduce the communication overheads between the mobile clients and the base stations for a transaction, and to alleviate the dependency of system performance on the performance of the underlying network. We propose to “*ship*” the entire transaction to the database server (base station) for processing instead of shipping every operation or data request to the database server. We call this approach as *transaction shipping*. Although the idea is simple, there exists many practical problems when it is applied to a MDRTDBS such as how to identify the execution path and the required data objects of a transaction before its execution, and how to deal with the dynamic properties in transaction execution. In here, we suggest a *pre-analysis* approach. The practicality of the pre-analysis comes from the observation that the *behavior of many real-time transactions is more predictable* comparing to the transactions in conventional database systems. To deal with the dynamic properties of the real-time transactions, a *data pre-fetching mechanism* is included in the transaction shipping approach to reduce the cost of incorrect prediction in the pre-analysis.

#### 4.2.1. Transaction Predictability

It is generally agreed that the functions and behavior of transactions in a real-time database application are more predictable. Mostly, they can be classified into different types and different transaction types will have different pre-defined behavior and critically [11]. For example in a medical information system, real-time transactions are for monitoring the physical status of a critical patient from various sensor devices, such as the blood pressure, the heart beat rate, and the body temperature. The arrival pattern and data requirements of the transactions are pre-defined. In some real-time database applications, e.g., programmed stock trading, although the transaction arrival pattern may be sporadic, their data requirements can be predicted with a high accuracy. For example, in a programmed stock trading, each investor may have a pre-defined investment plan, e.g. what their interested stocks are and how to make the trading analysis under different conditions. In a traffic navigation system, the physical connections of the roads are pre-defined. When searching the best path to a

destination from the current position based on the current road conditions, the set of roads to be search is pre-defined.

#### 4.2.2. Pre-analysis Phase

In the transaction shipping approach, the execution of a transaction is divided into two phases: the *pre-analysis phase* and the *execution phase*. Figure 2 shows the transaction architecture when it is processed under the transaction shipping approach. Once a transaction is initiated at a mobile client, a coordinator process, called *master coordinator*, will be created at the mobile client. Before shipping a transaction to the base station, the system will perform a pre-analysis on the transaction to derive its characteristics, e.g., what the operations of the transaction are and what the execution path of the transaction is. The concept of pre-analysis is similar to the two phase methods discussed in [23]. However, it should be noted that [23] is concerned about how to reduce the unpredictability in data access by using the concept of access invariant. In here, we use the pre-analysis to predefine the execution path of a transaction in order to reduce the number of communications between the mobile clients and the base stations for a transaction.

The pre-analysis of a transaction consists of two phases. In the first phase, the *set of operations* in the transaction will be identified. It is usually not difficult to identify the operations in a transaction. For example, if SQL statements are used to access the database, the SELECT statements are read operations and the INSERT statements are write operations. Note that transactions are assumed to have a simple flat structure. At this stage, it may not be necessary to identify the set of data objects required by the operations. Actually, it may not be easily done at the mobile client as it only contains limited information about the database system and the location of the data objects in the system. The required data objects of an operation will be determined while the transaction is executing at the base stations.

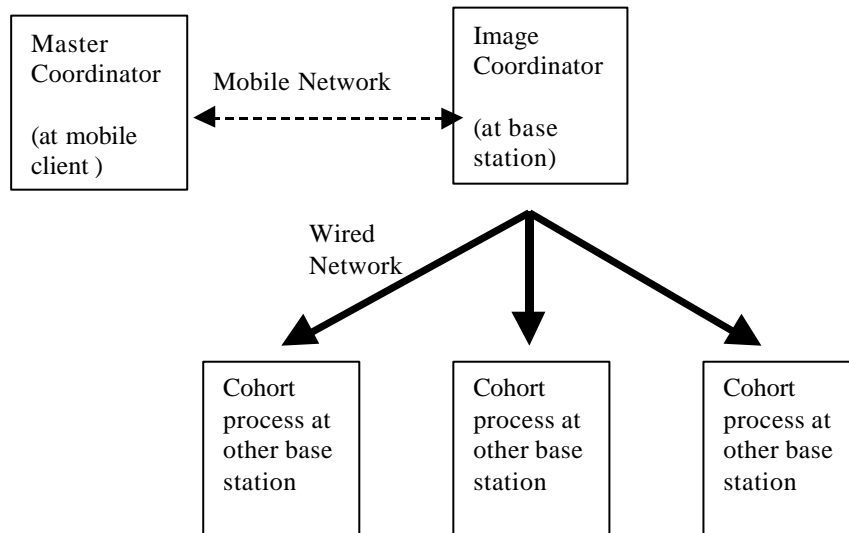


Fig. 2: Process Architecture under Transaction Shipping Approach

In the second phase, the *execution path* of the transaction, e.g., the precedence relationships of the operations, will be determined. For some transactions, the whole execution path cannot be determined until the data objects required by the transactions have been identified. For example, some conditional statements are based on the values of the data objects. For such transactions, the pre-analysis may identify the transaction type first and then make the prediction based on the pre-defined characteristics of that transaction type.

After the completion of the pre-analysis, a signature of the transaction will be created. A signature transaction  $S_i$  for transaction  $T_i$  consists of a 4-tuple:

$$S_i = (O_i, D_i, C_i, \prec_i)$$

where  $D_i$  is the deadline of  $T_i$ .

$C_i$  is the criticality of  $T_i$ .

$O_i$  is a subset of the operations in  $T_i$ .

$\prec_i$  is the partial order relationship among operations in  $O_i$ .

$<_i$  defines the precedence relationship among the operations. If  $Op_j <_i Op_k$ , then  $Op_k$  can start its execution only after the completion of  $Op_j$ , where  $Op_j$  and  $Op_k$  are operations of  $T_i$ .

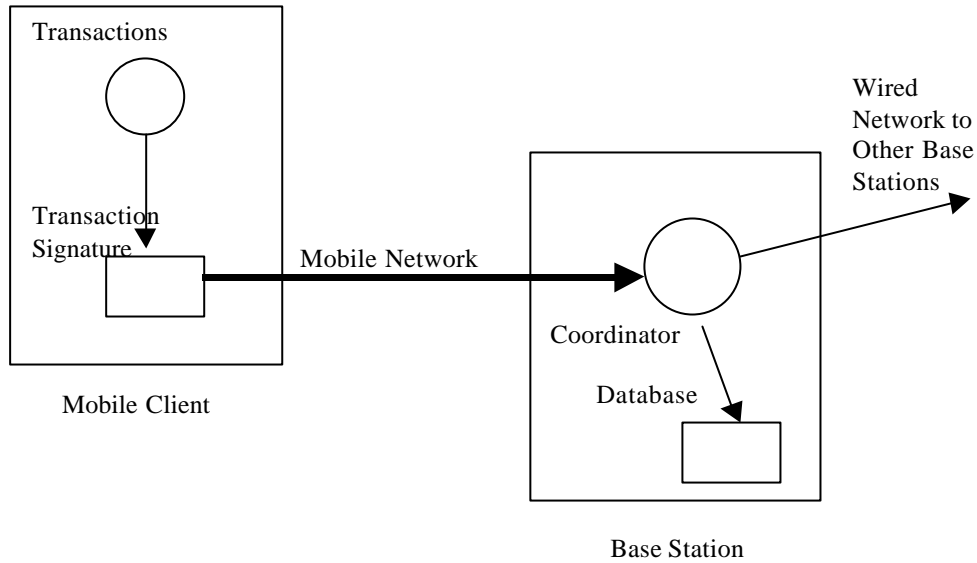


Fig. 3: The Transaction Shipping Approach

#### 4.2.2. Execution Phase

The signature transaction is forwarded to the base station of the MC through the mobile network. Once the server at the base station receives the transaction signature, it will create a process, called *image coordinator*, for the transaction. The image coordinator will take over the job from the master coordinator to process the transaction. Other transaction processes (cohorts) for the executions of the transaction will be created at other base stations if the operations of the transaction require to access the data objects located at that base stations.

The benefit of defining an image coordinator at the base station is that the connection between base stations is much better than the connection between the MCs and its base stations. Thus, it facilitates the management of the transactions and improves the performance of the atomic commitment protocol. Whenever a transaction has to be restarted, all its cohort processes (excluding the master coordinator and the image coordinator) will be destroyed after the completion of undo operations. The image coordinator is responsible for restarting the transaction from its beginning if its deadline has not been missed.

#### 4.2.3. Dynamic Properties of Transactions

Although the transactions in real-time database applications are more well-defined, they may still have some dynamic properties. Due to the dynamic properties and the interactivities between transactions and mobile users, data input from MC may still be needed while a transaction is executing. As a result, the pre-analysis of a transaction may need to be re-done while it is executing. For that case it has to go back to its originating MC. Before a transaction goes back to the mobile client, the system will pre-fetch the data objects possibly needed by the next operation of the transaction, and then ships the data objects with the transaction back to the mobile client. The purpose of the *data pre-fetching mechanism* is to process the next operation of the transaction at the mobile client so that the total number of communications between the mobile client and the base station can be reduced.

The assumption of the data pre-fetching mechanism is that the data requirements of the next operation can be predicted with a high accuracy. Again, the validity of the assumption is based on the well-defined behavior of real-time transactions. Suppose the execution path of a transaction  $T_1$  is originally predicted to be Path 1 (operations:  $Op_1$ ,  $Op_2$  and then  $Op_3$ ) in the first pre-analysis as shown in Figure 4. After the completion of operation  $Op_2$ , it finds out that the original prediction is incorrect, and  $T_1$  has to go back to the mobile client to “re-define” the execution path which is either Path 2 or Path 3 according to the pre-defined behavior of the transaction. Before,  $T_1$  is shipped back to the mobile client, the system pre-fetches the data objects required by  $Op_4$  and  $Op_5$ . Later at the mobile client,  $T_1$  can process the next operation  $Op_4$  or  $Op_5$ . After that,  $T_1$  is sent to the base station to commit or continue its following operations. Note that under the data pre-fetching mechanism,  $T_1$

has to set a lock on the pre-fetched data objects. The purpose to is to ensure the serializability of transaction executions.

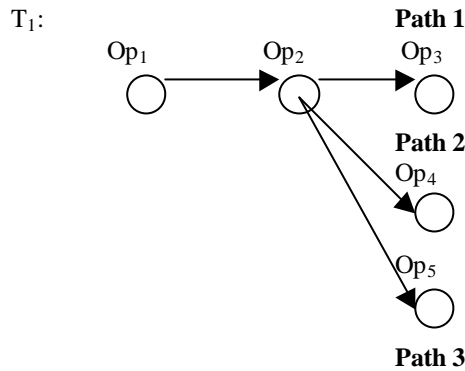


Fig. 4: Data Pre-fetching Mechanism

Note that the transaction shipping approach does not require the transmission of a large number of data objects to the clients as required in the data shipping approach. This is an attractive feature to MDRTDBS as the communication bandwidth is low. Although some data objects are required to be shipped with a transaction in the data pre-fetching mechanism, the data volume should be much smaller than in the data shipping approach as the data pre-fetching mechanism is required only when the predicted execution path is incorrect. Furthermore, only those data objects, which are required by the next operation, are shipped.

It is obvious that the performance gained from the pre-analysis depends on its accuracy and the dynamic properties of the transactions. In the best case, one single communication between the MC and the base station is required for transaction execution. In the worst case, the number of communications between the mobile client and the base station for each transaction execution is equal to the number of operations in the transaction. If the system can make a good estimation in the pre-analysis, a lot of communications can be saved.

## 5. SIMILARITY-BASED DISTRIBUTED HIGH PRIORITY TWO PHASE LOCKING

Restarting transactions in a MDRTDBS can be very expensive and the restarted transactions will have a high probability of missing their deadlines. To reduce the probability of data conflicts and transaction restarts, a less restrictive notion of correctness criterion may be explored for concurrency control, e.g., [7, 12, 15, 26]. In particular, the concept of *similarity*, which is based on the time-validity of data, is shown effective for RTDBS [12, 13]. In this paper, we shall consider the technical issues in the design of a similarity-based real-time locking protocol for MDRTDBS. We will incorporate the concept of similarity into the DHP-2PL with an attempt to further increase the system concurrency, and to reduce the number of transaction restart and blocking due to data conflicts. The new protocol is called *Similarity-Based Distributed High Priority Two Phase Locking* (SDHP-2PL).

Although the concept of *similarity* for concurrency control in RTDBS is not new, many technical issues in the design of a similarity-based real-time locking protocol are still not clear such as how to resolve the lock conflicts between the read and write operations when some of the conflicts are similar and some of them are not. Furthermore, it will be interesting to see what will be the effectiveness of using similarity in improving the performance of a MDRTDBS.

### 5.1. The Basic Strategies

#### 5.1.1. Data Similarity

*Similarity* is closely related to the important idea of imprecise computation in real-time systems [19] and to the idea of partial computation for databases [4]. It has been shown to be very effective in improving the performance of RTDBS. For many real-time applications, the value of a data object that models an object in the external environment cannot, in general, be updated continuously to perfectly track the dynamics of the external object. At the same time, it is also *unnecessary* for data values to be perfectly up-to-date or precise to be useful. In particular, the data values of a data object that are slightly different are often interchangeable as read data for transactions. This observation underlies the concept of *similarity* among data values.

The concept of *similarity* can be used to extend the conventional correctness criterion for concurrency control in RTDBS [12, 13], and to balance data precision (based on *similarity*) and system workload [7]. The notions of *similarity* and *strong similarity* were originally introduced in [12], which have the property that

swapping similar conflicting operations in a schedule will always preserve *similarity* in the output. In other words, if two operations in a schedule are strongly similar (i.e., they are either both writes or both reads, and the two data values involved are strongly similar), then they can always be used interchangeably in a schedule without violating the integrity and consistency of the database. For the rest of this paper, unless explicitly specified, all *similarity* relations considered in this paper are strong *similarity* relations, and all similar data can be used interchangeably in a schedule without adverse results. We refer interested readers to [12, 13] for the justification of schedule correctness.

### 5.1.2. The Basic Strategies for Conflict Resolution

Specifically, we assume that the application semantics allow us to derive similarity bounds for some of the data objects such that two write operations on the data object must be similar if their time-stamps differ by an amount no greater than the similarity bound of the data object, e.g., all write operations on the same object that occur in any interval shorter than the similarity bound of the data object can be swapped in the (untimed) schedule without violating consistency requirement [13]. The time-stamp of a data object indicates at which snapshot the current value of the data object is taken from the external environment. Different data objects may have different similarity bounds. For example, the transactions in a stock trading system may consider the values of the last traded price of a stock to be similar if they are updated within a time frame of 10 seconds. For a warehouse keeping system, the similarity bound for the inventory data may be in minutes.

The existence of similarity bounds provides more freedom in concurrency control: whereas two conflicting transactions must be totally ordered to preserve serializability, they need not be if their conflicting operations occur in an interval shorter than the similarity bound, and can therefore be executed concurrently. The basic strategy for conflict resolution in the Similarity-Based Distributed High Priority Two Phase Locking (SDHP-2PL) protocol can be summarized as follows:

- (i) Suppose transaction  $T_i$  issue a *read-lock request* on data object  $D_k$ , and  $D_k$  is already write-locked by a collection of transactions TH. Let  $T_j$  be the most recently committed transaction which updates  $D_k$ . If the write operation (on  $D_k$ ) of  $T_j$  and the write operations (on  $D_k$ ) of *all* transactions in TH are *similar*, then the read-lock request on data object  $D_k$  may be granted. It is because  $T_i$  will always read from *similar* data values, regardless of which transaction in TH commits (or updates  $D_k$ ) first.
- (ii) Suppose transaction  $T_i$  issue a *write-lock request* on data object  $D_k$ , and  $D_k$  is already write-locked by a collection of transactions TW and read-locked by a collection of transactions TR. Let  $T_j$  be the most recently committed transaction which updates  $D_k$ . There are two cases to consider:
  - (1) If TR is empty, and the conflicting write operation of  $T_i$  and the conflicting write operations of all transactions in TW are similar, then the write-lock of  $T_i$  may be granted. It is because the final state of the database will always be *similar* regardless of which write operation performs first.
  - (2) If TR is not empty, and the conflicting write operation of  $T_i$ , the conflicting write operation of  $T_j$ , and the conflicting write operations of all transactions in TW are *similar*, then the write-lock of  $T_i$  may be granted. It is because the final state of the database will always be *similar* and the read operations will read from *similar* data value regardless of which write operation performs first.

Obviously, if the above *similarity* test of a lock request fails, then two alternatives must be considered:

- (1) the lock-requesting transaction may be blocked; and
- (2) the transactions, which cause the blocking, may be restarted.

Which alternative should be used depends on the priorities of the conflicting transactions. In the SDHP-2PL, we adopt both policies as follows:

Let CT be a subset of TH which consists of transactions that lock  $D_k$  in a mode conflicting and non-similar to the lock request of  $T_i$ , and SCT be a subset of CT which consists of transactions with priorities lower than  $T_i$ . If the restart of all transactions in CT will make  $T_i$  passing the *similarity* test and the priorities of all the transactions in CT are lower than  $T_i$ , then the transactions in CT are restarted, and the lock request of  $T_i$  is granted.

If some transactions in CT have priorities higher than  $T_i$ , i.e., SCT is not equal to CT, simply restarting all the transactions in SCT cannot resolve the lock conflict problem. On the other hand, restarting all the transactions in CT including the higher-priority transactions may have the cyclic restart problem as the restarted higher-priority transactions may restart  $T_i$  later.

There are two possible solutions to resolve the above problem. Firstly, we can use an *aggressive approach* in which  $T_i$  will be blocked when CT is not equal to SCT and all the transactions in SCT are restarted. The reason of restarting the transactions in SCT is to minimize the blocking time of  $T_i$  as the time that required

to complete the lower-priority transactions can be very long especially in an unreliable mobile network. Whenever a transaction in (CT – SCT) releases its lock on  $D_k$ , a similarity test will be performed for  $T_i$ . In this way, sooner or later,  $T_i$  will set a lock on  $D_k$  when all the higher-priority transactions in the (modified) CT have released their locks on  $D_k$ . To prevent repeat restarts of lower-priority transactions, a transaction is not allowed to set a lock if a higher-priority transaction, which is non-similar to it, is waiting to set the lock.

In the aggressive approach, by restarting the lower-priority transactions, the blocking time of the higher-priority transaction can be minimized. However, the cost is more transaction restarts and this is not desirable in a mobile environment as a restarted transaction will have a higher probability of deadline missing. The second method is a *conservative approach*, in which the lock-requesting transaction will be blocked if (CT – SCT) is not empty, e.g., there exists a higher-priority transaction in CT. Each time when a transaction releases a lock on  $D_k$ , a similarity test will be performed for  $T_i$ .  $T_i$  will be allowed to set a lock on  $D_k$  when all the higher-priority lock-holding transactions are similar to it. The number of transaction restarts under the conservative approach should be smaller than the aggressive approach. However, the blocking time of a higher-priority transaction may be longer and it also has the indefinite postponement problem. In the performance experiments (in Section 6), we will investigate the relative performance of these two approaches for MDRTDBS.

Note that we propose to use similarity bounds to quantify the similarity of conflicting operations in this paper. However, we should emphasize that the similarity test in the SDHP-2PL can also be re-implemented as a test for data-value similarity. For example, let transaction  $T_i$  issues a write-lock request on data object  $D_k$ . Under the data-value-based similarity test, the write-lock request must be associated with an intended written value  $V_i$ . The data-value-based similarity test must compare  $V_i$  with the intended written values of executing transactions and the current value of  $D_k$  to determine whether the lock request of  $T_i$  should be granted. The lock request is granted only if all of the values are similar. The choice between a similarity-bound-based similarity test and a data-value-based similarity test in a MDRTDBS is entirely application-dependent. For the purpose of this paper, we should not focus the discussions of SDHP-2PL on the selection of similarity tests. Instead, we should explore the effectiveness of using similarity for MDRTDBS.

## 5.2. SDHP-2PL Protocol Definition

In this section, we define the *Similarity-Based Distributed High Priority Two Phase Locking (SDHP-2PL)* following the basic strategies defined in the last sub-section.

Let a transaction  $T_r$  request a lock in a specified mode, i.e., read or write, on a data object  $D_k$  residing at a base station.  $T_r$  invokes the lock request procedure of the SDHP-2PL, which is defined, as follows, to set the lock.

```

Lock-Request ( $T_r$ , Mode,  $D_k$ )
  Begin
    If the lock request of  $T_r$  is similar to existing locks on data object  $D_k$ 
      Then /* Please see Section 5.1.2 for the similarity of locks */
        Return(Success)
    Else
      Let CT be the set of transactions that lock  $D_k$  in a mode
        conflicting and non-similar to the lock request of  $T_r$  and
        SCT be a subset of CT which consists of transactions with a priority lower than  $T_r$ 
      Case Aggressive approach
        For each transaction  $T_h$  in SCT Do
          If  $T_h$  is not committing
            Then
              Restart  $T_h$  locally (or globally) if  $T_h$  is a local
                (or global) transaction.
              Remove  $T_h$  from CT
            Else
              Priority( $T_h$ ) = Priority( $T_r$ ) + fixed priority level
          EndIf
        EndFor
        If CT is still not empty
          Then
            Block  $T_r$  until all transactions in CT release conflicting and
              non-similar locks.
          EndIf
        Return(Success)
      Case Conservative approach
        If CT is not empty and CT  $\neq$  SCT
          Then
            Block  $T_r$  until all transactions in (CT-SCT) release conflicting and
  
```

```

        non-similar locks.
    Else
        For each transaction  $T_h$  in CT Do
            If  $T_h$  is not committing
                Then
                    Restart  $T_h$  locally (or globally) if  $T_h$  is a local
                    (or global) transaction.
                    Remove  $T_h$  from CT
                Else
                    Priority( $T_h$ ) = Priority( $T_r$ ) + fixed priority level
                EndIf
            EndFor
        EndIf
    Return(Success)
EndIf
End

```

### 5.3. Implementation and Performance Issues

The implementation of the similarity-based protocol will not incur heavy additional overheads. The main overhead is in the checking of similarity. The checking is performed whenever a lock conflict has occurred. Since the lock table is assumed at the main memory, the comparison amongst the similarity bounds of the conflicting data objects and the time-stamps of the conflicting transactions should be able to be performed in an efficient way. On the other hand, the benefit of a similarity based concurrency control protocol is an increase in system concurrency. If the similarity bounds of the data objects are well-chosen, the benefit obtained from an increased in concurrency control should be more than the overhead for the similarity checking.

As astute reader may notice that an increase in concurrency may not necessarily result in better performance, e.g., higher probability of meeting the transaction deadlines. If the similarity bounds are small, even though some of the lock conflicts are similar and concurrent accesses of a data object from different transactions are allowed, the transactions may later be restarted due to a conflict with another higher-priority transaction which is not similar to the executing transactions. The consequence is a greater cost for resolving the lock conflict as more transactions are restarted and greater amounts of resources are wasted. Thus, the effectiveness of the similarity approach is highly dependent on the values of the similar bounds of the data objects relative to the time required to complete a transaction. Since mobile network is slow and unpredictable, the time required to complete a transaction can be very long. It may be that only large similar bounds can give a significant improvement to the system performance. The effectiveness of similarity for concurrency control in MDRTDBS and the performance of the SDHP-2PL will be studied in the next section.

## 6. PERFORMANCE EXPERIMENTS

We have developed a simulation program for the MDRTDBS model introduced in Section 2 and simulation experiments are performed:

- (1) to study the performance of the DHP-2PL as compared with the HP-2PL in a MDRTDBS;
- (2) to study the performance of the transaction shipping approach, as compared with the query shipping approach;
- (3) to compare the performance of the two approaches, aggressive and conservative, for resolving data conflicts in the SDHP-2PL and to identify the effectiveness of using *similarity* for concurrency control in a MDRTDBS.

Note that the purposes of the simulation studies are not to investigate the performance of the proposed protocols and approaches at a specific mobile environment and for a specific real-time database application. Instead, the objectives are to identify the performance characteristics of the protocols and approaches, and to demonstrate the capability of the algorithms in improving the performance of MDRTDBS. In the experiments, we shall focus ourselves on the issues related to mobile network and data conflict resolution.

### 6.1. Simulation Model

A simulation program has been developed using OpNET, which is a proprietary simulation tool according to the MDRTDBS model introduced in Section 2. In OpNET, a radio module is provided for mobile communication [22]. With the radio module, most of the details of a cellular radio network, such as the mobile clients, MTSO and base stations are implemented and the unique features of a mobile network such as call-setup procedure, tear down features, and handoff procedure, are modeled explicitly. In the simulation program, the

clients are modeled with mobility and may move around and even cross the cell borders. Different clients are given different trajectories. They can self-locate themselves based on the received signal strength from the base stations and communicate with the base stations using pre-defined radio signals via the uplink and downlink channels. Disconnection is modeled explicitly by defining a probability for a disconnection between a mobile client and its base station every time when a channel request is made. Transient errors in communication are modeled by a noise factor, which affects the strength of radio signal received by the base stations and mobile clients.

In the system level, a DRTDBS model is implemented [31] in which the Distributed High Priority Two Phase Locking (DHP-2PL) is used for concurrency control. The database system at each base station is shown in Figure 5. It consists of a scheduler, a CPU, a ready queue, a local database, a lock table, and a block queue. It is assumed that the database is resided at the main memory in order to eliminate the impacts of disk I/O scheduling on the system performance [1, 33]. To simplify the model, we further assume that all the temporal data objects have the same *avi* and *rvi*.<sup>4</sup> In each local database system, an update generator creates updates periodically to refresh the validity of the temporal data objects. Updates are single operation transactions. They do not have deadlines. The priorities of the updates are assigned to be higher than the priorities of all of the other transactions in order to maintain the validity of the temporal data objects [36].

Transactions are generated from mobile clients sporadically. Each transaction is defined as a sequence of operations. It is assumed that the transactions have the same criticality level. Let each operation access a single data object, and the required data object of an operation is evenly distributed in the database. The operations have similar CPU requirement statistically. Transactions wait in the ready queue for CPU allocation, and the CPU is scheduled according to the transaction priorities which are assigned based on the earliest deadline first policy. Since we assume that the transactions are associated with firm deadlines, the scheduler will check the deadline before a transaction is allocated the CPU. If the deadline is missed, the transaction is aborted immediately. If any of the temporal data objects accessed by the transaction becomes invalid before the commitment of the transaction, the transaction is aborted as it may observe some out-dated data values. After the completion of a transaction, the mobile client will generate another transaction after a think time.

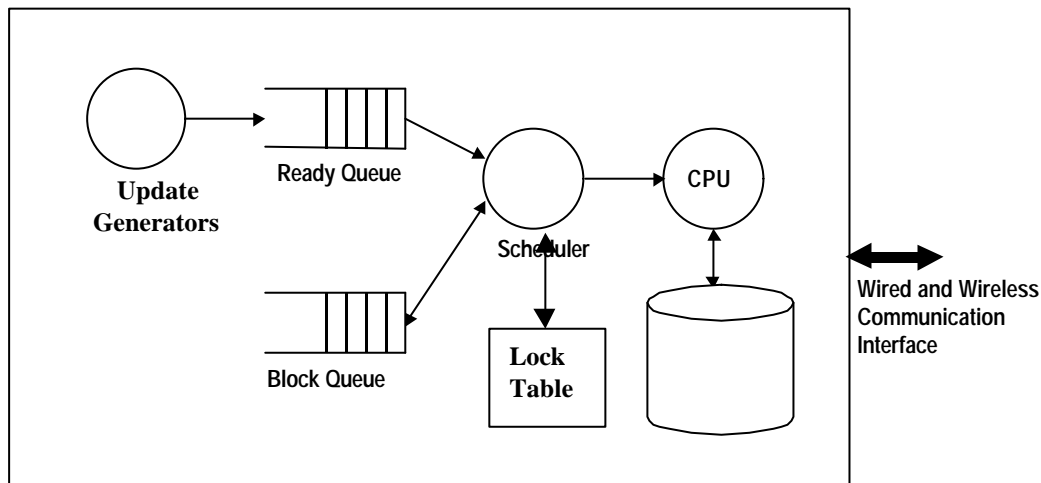


Fig. 5: Model of a database system in a base station

In the model, the Distributed High Priority Two Phase Locking (DHP-2PL) is employed for concurrency control. (In the second set of experiments, we will replace it with the SDHP-2PL.) A lock table is maintained for the data objects residing at each base station. The scheduler detects lock conflicts by examining the lock table. Before an operation is processed, its required lock has to be set in an appropriate mode. After the completion of all the operations of a transaction, the transaction enters the commit phase in which the two phase commit protocol will be performed. The locks of a transaction will be released upon its commitment.

## 6.2. Model Parameters and Performance Measures

Similar to many previous studies on single-site and distributed RTDBS, the deadline of a transaction,  $T$ , is defined according to the expected execution time of a transaction [1, 3, 8, 8, 17, 18]:

<sup>4</sup> Note that if the *avi* and *rvi* or the temporal data objects are different and some of them have a smaller value, the probability of transaction abort will be higher due to tighter temporal constraint. However, the impact of this factor on the relative performance of the proposed protocols and approaches should be similar.

$$Deadline = ar(T) + pex(T) \times (1 + SF)$$

where  $SF$  : the slack factor which is a random variable uniformly chosen from a slack range;  
 $ar(T)$ : the arrival time of transaction  $T$ ;  
 $pex(T)$ : the predicted execution time of  $T$ . It is defined as:

$$pex(T) = (T_{lock} + T_{process} + T_{update}) \cdot N_{oper}$$

where  $N_{oper}$ : the number of operations in the transaction;  
 $T_{lock}$ : the CPU time required to set a lock;  
 $T_{process}$ : the CPU time required to process an operation; and  
 $T_{update}$ : the CPU time to update a data object (for write operations) .

Since different transaction processing strategies will expect different transaction execution times, we assume that all the operations of the transactions are local operations in the calculation of transaction deadlines. The baseline setting of the model is shown as follows:

Parameters	Baseline Values
<b>System Level</b>	
Number of MTSO	1
Number of Cell Sites	7
Location Update Interval	0.2 second
Transmission Speed for Channel	10 kbps
Number of Channels for Each Cell Site	10
<b>Transaction</b>	
Think Time	4 seconds
Transaction Size	7 to 14 operations, uniform distribution
Proportion of write operations	1.0
Slack range	10 – 20 (the slack factor is uniformly distributed in the slack range)
<b>Mobile Network</b>	
Number of Mobile Clients	84
Channel Connection time (CL)	1 second
Call Update Interval	0.2 second
Disconnection probability	0.5%
<b>Database</b>	
Number of Local Databases	7 (1 in each base station)
Database Size	200 data objects per local database
Concurrency Control	Distributed High Priority Two Phase Locking (DHP-2PL)
Fraction of Temporal Data Objects	10%
Temporal Data Object Update Interval	0.5 update per second per data object
Absolute Threshold (avi)	12 seconds
Relative Threshold (rvi)	8 seconds
<b>CPU</b>	
CPU Scheduling	Earliest Deadline First
CPU time to process an operation	34 ms
CPU time to set a lock	1 ms
CPU time to release a lock	1 ms
CPU time to check a lock	1 ms
CPU time to update a data object	6 ms
CPU time for pre-analysis	100ms
Deadline Missing Treatment	Firm deadline, abort the transaction once the transaction deadline is found missing

Table 1: Model parameters and their baseline values

The channel connection time (CL) is the time required to send a message from an MC to its base station. It is defined based on the transmission speed of the channel and the message size. For a transmission speed of 10kbps, the channel connection time is 1 second for a message of 1Kbytes. The location update interval must be smaller than the channel connection time. Otherwise, a channel will not be able to be created after an MC has crossed the border into another cell site.

In the simulation program, the locality of transactions in accessing data objects is not modeled explicitly. The reason is that data locality is application-dependent, and it is not our objective to study the performance of the system for a specific application. Instead, a small database is used which allows us to study the effect of hot-spots, in which a small part of the database is accessed very frequently by most of the transactions. Another benefit of using a small database is to create a high data contention environment. It helps us to understand the performance characteristics of the concurrency control protocol. A small database means that the degree of data contention in the system can be easily controlled by the sizes of the transactions.

The primary performance measure used is the *miss rate*. It is defined as the number of transactions which miss deadlines over the total number of transactions generated. In addition to miss rate, we also measure the *conflict probability* (CP) which is defined as the total number of lock conflicts over the total number of lock requests. The conflict probability can be used as an indicator of the degree of data contention in the system. Other measures are *CPU utilization* and the *abort with restart probability* (ARP). The CPU utilization gives the degree of resource contention at the base station. The ARP is defined as the number of aborted transactions, which had been restarted due to lock conflict, over the total number of transactions which have been restarted. It can be an indicator of the probability of deadline missing due to transaction restart.

### 6.3. Simulation Results and Discussions

In the following sub-sections, we report the important simulation results obtained from the simulation experiments. In each simulation run, the simulation time is 1,000sec. For a think time of 4 seconds, about 10,000 transactions are generated from the 84 mobile clients in each simulation run. The length of the simulation is determined after a number of trial runs using different simulation lengths until stable results are obtained.

#### 6.3.1. Performance of the DHP-2PL

In this set of experiments, we compare the performance of the DHP-2PL with the distributed version of HP-2PL in which the lock-holding transaction will be restarted when the priority of the lock-requesting transaction is higher.

In Figure 6, we can see that the performance of the DHP-2PL is consistently better than the HP-2PL at different transaction workloads. An increase in think time decreases the transaction workload. The better performance of the DHP-2PL is due to its specific lock conflict resolution mechanisms for mobile network: (1) priority inheritance for resolving the lock conflict where the lock holding transaction is committing; and (2) cautious waiting for resolving the lock conflict where the lock holding transaction is suspected to be a disconnected transaction. Although the probability of network disconnection is very low as defined in the experiments, e.g., 0.5%, its impacts on system performance can be very significant due to the chain of blocking effect. A disconnected transaction can block several transactions directly and transitively. Consequently, all the blocked transactions may miss their deadlines. The higher lock conflict probability can be observed in Figure 7 in which we can see that the conflict probability of the DHP-2PL is consistently lower than that of the HP-2PL. We repeat the experiments for a system where the deadline constraints of the transactions are tighter, e.g., the slack range is 20 to 25. The results are shown in Figure 8. Consistent with the results in Figure 6, the miss rate of the DHP-2PL is significantly lower than that of the HP-2PL for different values of think time.

#### 6.3.2. Performance of the Transaction Shipping Approach

In this set of experiments, we study the performance of the transaction shipping approach (TS) as compared with the query shipping approach (QS) at different return probabilities. The reason of not comparing with the data shipping approach is that the data shipping approach may require the transmission of large amount of data and is not suitable to MDRTDBS. Under the transaction shipping approach, a transaction has to go back to its originating mobile client while it is executing when: (1) it has to receive input data from the mobile client; or (2) the prediction at the pre-analysis phase is incorrect. For both cases, the pre-analysis for the transaction has to be performed again at the mobile client.

In the experiments, the *return probability* defines the probability for which a transaction has to go back to its originating mobile client after the completion of an operation. The value of return probability depends on the accuracy of the prediction obtained at the pre-analysis phase and the dynamic behavior of the transactions.

When the return probability is set to be zero, it means that the prediction is correct, and the transaction does not need to go back to the mobile client once it has been transmitted to the base station. If the return probability is set to be one, the prediction completely fails. Every time after an operation has been completed, the transaction has to go back to the base station to refine the execution path for its remaining operations.

Figure 9 shows the miss rates of the transactions shipping approach (TS) and the query shipping approach (QS) at different return probabilities and when the channel connection times are 1 second ( $CL = 1$ ) and 2 seconds ( $CL = 2$ ). From the figure, it can be seen that the performance of the system is greatly improved with the use of the transaction shipping approach, especially when the return probability is low (a high accuracy of prediction in the pre-analysis phase). This is consistent with our expectation. Under the transaction shipping approach, the number of communications required to process a transaction is much reduced. Thus, the transactions can be completed earlier as the total mobile network delay is much reduced. Even though a transaction has been restarted due to a lock conflict with a higher-priority transaction, the restarted transaction still has a high probability to be completed before its deadline. Thus, the abort with restart probability (ARP) is much lower under the transaction shipping approach than under the query shipping approach as shown in Figure 10. With the use of transaction shipping approach, the workload in the CPU is generally higher as some of the workload on the network is now shifted to the base stations. Thus, we can see that the CPU utilization increases with a reduction in the return probability as shown in Figure 11.

An important observation in Figure 9 is that even though the return probability is 1, e.g., the prediction is wrong every time, the performance of the transaction shipping approach (TS) is still much better than the query shipping approach (QS). It is due to the data pre-fetching mechanism in the transaction shipping approach. Even though a wrong prediction has been made in the pre-analysis phase, the system still can pre-fetch the required data of the next operation of the transaction based on the pre-defined characteristics of the transaction. Thus, the number of communications between the base stations and mobile clients is still much reduced.

As also can be observed in Figure 9, the improvement of using the transaction shipping approach is smaller when the channel contention time is long and the return probability is high, e.g.,  $CL = 2$  and return probability = 1. The reason is that at a higher channel contention, even with the use of the transaction shipping approach, the transactions may still have a high probability of missing deadlines. As shown in Figure 10, the abort with restart rate for the transaction shipping approach is higher at longer channel connection time.

We repeat the experiments using a looser deadline constraint for the transactions, e.g., slack range is 20 – 25. The results are shown in Figure 12 to Figure 14. Consistent with the previous set of results, the use of transaction shipping approach still can greatly improve the system performance when the transactions have looser deadline constraints, e.g., more slack time for execution.

Figure 15 shows the impacts of different pre-analysis overheads (in terms of the amount of time) on the miss rate under the transaction shipping approach when the return probabilities (RR) are 0.8 and 1.0. One interesting observation is that the overheads do not have any significant effect on the performance of the transaction shipping approach. Although the overheads may make the deadlines tighter, because of the lengthened time required to complete a transaction, it also releases the degree of resource contention in the system (the mobile network and the base stations), especially on the channels. It is because the transactions now spend more time at the mobile client side, instead of at the mobile network and base stations.

### 6.3.3. Performance of the Similarity-based Protocols

Figure 16 shows the performance of the SDHP-2PL using the aggressive approach (SDHP-2PL-Ag) and the conservative approach (SDHP-2PL-Con) at different similarity bounds. It is surprise to see that for both SDHP-2PL-Ag and SDHP-2PL-Con relaxing the correctness of concurrency control to similarity may not always improve the system performance. If the similarity bound is very small, e.g.,  $\leq 1$  seconds, the miss rates are even higher than the case where the similarity bound is zero, e.g. the protocol is reduced to the DHP-2PL. The main reasons may be that:

- (1) although relaxing the correctness criterion to similarity can increase the system concurrency, the number of transaction restarts may be higher if the similarity bound is very small. More than one transaction may be allowed to access a data items concurrency if they can pass the similarity test. However, all of them may be restarted if they are not similar to a higher-priority transaction.
- (2) The transaction restart overhead is very high in a mobile environment. The probability of deadline missing will be much higher if a transaction is restarted.

As shown in Figure 16, the values of the similarity bounds of the data objects play an important role in the effectiveness of using similarity for concurrency control. A significant amount of improvement is achieved for larger similarity bounds, e.g., similarity bound  $\geq 2$  seconds. As expected, the improvement is due to smaller blocking and restart probability. Smaller conflict probabilities can be observed in Figure 17 in which the conflict probability of both SDHP-2PL-Ag and SDHP-2PL-Con decreases with an increase in similarity bound.

When we compare the performance of the SDHP-2PL-Ag with SDHP-2PL-Con, we can see that in general SDHP-2PL-Con gives a better performance although their performance is very similar especially when the similarity bound is very small and very large. The main reason of the slightly poor performance of the SDHP-2PL-Ag is due to the heavy restart overhead. In the SDHP-2PL-Ag, a lower-priority transaction will be restarted if the priority of the lock requesting transaction higher. The restarted transaction will have a high probability of miss its deadline. As shown in Figure 18, the abort with restart probability is higher in the SDHP-2PL-Ag than that in the SDHP-2PL-Con. When the similar bound is large, the performance of SDHP-2PL-Ag and SDHP-2PL-Con becomes very similar as most of the conflicts are resolved by similarity.

## 7. CONCLUSIONS

The design of mobile distributed real-time database systems (MDRTDBS) is receiving growing interests in recent years. Due to the poor quality of services provided by a mobile network, it is not easy to meet the deadlines of the transactions in a MDRTDBS. In this paper, we define a detailed model for MDRTDBS, in which the mobility of the mobile clients and characteristics of the mobile network, e.g., disconnection and low bandwidth, are modeled explicitly. We have designed a distributed real-time locking protocol, called Distributed High Priority Two Phase Locking (DHP-2PL), where the characteristics of the mobile network are considered in resolving the conflicts in data accesses. Then, we propose two strategies to improve the system performance and to reduce the impact of mobile network on the performance of the adopted concurrency control protocol. We first propose the concept of transaction shipping to reduce the dependency of a concurrency control protocol on the performance of the underlying network. With the transaction shipping approach, the communication overheads for processing a transaction can be much reduced. A data pre-fetching mechanism is included in the transaction shipping approach to deal with the dynamic properties of transactions and inaccuracy of prediction in the pre-analysis. We then adopt the notion of similarity to resolve conflicts among data access that can be very costly over a mobile network. Different issues in the design of similarity-based real-time locking protocol are discussed. In the design of similarity-based locking protocol, special attention should be paid in resolving a lock conflict in which some of the lock holders are similar to the lock requester while some of them are not. Two methods, the aggressive and conservative approaches, are suggested to resolve the conflicts.

Simulation experiments have been conducted to investigate the performance of the DHP-2PL protocol, the effectiveness of the transaction shipping approach and the similarity-based protocols. With the transaction shipping approach, the number of deadline violations is greatly reduced as the contention for channels, the time spent on communication, the probability of lock conflict, and the amount of resources wasted on restarted transactions are much reduced. The transaction shipping approach can also help balance the workload in the system (between the channels and the base stations). The use of similarity-based algorithm further improves the system performance by reducing the number of lock conflicts. However, the experimental results show that the effectiveness of similarity depends very much on the values of the similarity bounds.

## ACKNOWLEDGEMENTS

The work reported in this paper was supported in part by research grants NSC87-2213-E194-002 of Taiwan National Science Council, the Hong Kong RGC CERG 9040353, Strategic Grants 700584 and 7000849 of City University of Hong Kong.

## REFERENCES

- [1] R.J. Abbott and Hector Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. *ACM Transactions on Database Systems* 17( 3): 513-560 (1992).
- [2] A. Datta, A. Celik, J. Kim and D.E. VanderMeer. Adaptive Broadcast Protocol to Support Power Conservant Retrieval by Mobile Users. *In Proceedings of 13th International Conference on Data Engineering*, pp. 124-133 (1997).
- [3] A. Datta, S. Mukherjee, P. Konana, I. Viguier, A. Bajaj. Multiclass Transaction Scheduling and Overload Management in Firm Real-Time Database Systems. *Information Systems*, 21(1):29-54 (1996).
- [4] S.B. Davidson and A. Watters. Partial Computation in Real-Time Database Systems. *In Proceedings of 5th Workshop on Real-time Software and Operating Systems* pp. 117-121, (1988).
- [5] M.J. Franklin. *Client Data Caching: A Foundation for High Performance Object Database Systems*, Kluwer Academic Publishers, Boston (1996).
- [6] J. Gray and A. Reuter. *Transaction Processing: Concept and Techniques*, Morgan Kaufmann (1993).
- [7] S.J. Ho, T.W. Kuo, A.K. Mok. Similarity-Based Load Adjustment for Real-Time Data-Intensive Applications. *In Proceedings of IEEE 18th Real-Time Systems Symposium*, pp.144-153 (1997).
- [8] J.R. Haritsa, M. Livny, M.J. Carey. On Being Optimistic about Real-Time Constraints. *In Proceedings of the 9<sup>th</sup> ACM Symposium on Principles of Database Systems*, pp. 313-343 (1990).

- [8] J. Huang, J. Stankovic and K. Ramamritham. Priority Inheritance in Soft Real-Time Databases. *Journal of Real-Time Systems*, 4(3):243-268 (1992).
- [9] T. Imielinski and B. R. Badrinath. Mobile Wireless Computing: Challenges in Data Management. *Communications of the ACM*, 37(10):18-28 (1994).
- [10] T. Imielinski and S. Viswanathan. Adaptive Wireless Information Systems. In *Proceedings of Special Interest Group on Database Systems (SIGDBS)*, Japan, pp. 19-41 (1994).
- [11] Y..K. Kim, S.H. Son. Supporting Predictability in Real-Time Database Systems. In *Proceedings of Real-Time Technology and Applications Symposium*, Brookline, Massachusetts, pp. 38-48 (1996).
- [12] T.W. Kuo and A.K. Mok. Application Semantics and Concurrency Control of Real-Time Data-Intensive Applications. In *Proceedings of IEEE 13<sup>th</sup> Real-Time Systems Symposium*, pp. 35-45 (1992).
- [13] T.W. Kuo and A.K. Mok. SSP: a Semantics-Based Protocol for Real-Time Data Access. In *Proceedings of IEEE 14<sup>th</sup> Real-Time Systems Symposium*, pp. 76-86 (1993).
- [14] E. Kayan and O. Ulusoy. An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems. *The Computer Journal*, 42( 6):501:510 (1999).
- [15] C. Lai. Optimistic Similarity-Based Real-Time Concurrency Control. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pp.189-198 (1998).
- [16] William C. Y. Lee, *Mobile Cellular Telecommunications Systems*, New York, McGraw-Hill (1989).
- [17] K W Lam, K Y Lam and S L Hung. Distributed Real-time Optimistic Concurrency Control Protocol. In *Proceedings of International Workshop on Parallel and Distributed Real-time Systems*, Hawaii, pp. 122-125 (1996).
- [18] Victor C.S. Lee, Kam-yiu Lam and Ben Kao. Priority Scheduling of Transactions in Distributed Real-time Databases. *Journal of Real-time Systems*, 15(1):31-61 (1998).
- [19] K.J. Lin, S. Natarajan, and J. W.-S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In *Proceedings of IEEE 8th Real-Time Systems Symposium*, pp. 210-217 (1987).
- [20] J. Lee and S.H. Son. Dynamic Adjustment of Serialization Order for Real-time Database Systems. In *Proceedings of 14<sup>th</sup> IEEE Real-time Systems Symposium*, North Carolina, pp. 66-75 (1993).
- [21] H.V. Leong and A. Si. Database Caching over the Air Storage. *The Computer Journal*, 40(7):401-415 (1997).
- [22] *OPNET Modeler/Radio 3.0.B*©, MIL 3, Inc. (1996).
- [23] P. O'Neil, K. Ramamritham and C. Pu. A Two-Phase Approach to Predictably Scheduling Real-time Transactions. In *Performance of Concurrency Control Mechanisms in Centralized Database Systems*, edited by V. Kumar, Prentice Hall, New Jersey (1996).
- [24] G. Ozsoyoglu and R. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):513-532 (1995).
- [25] E. Pitoura and B. Bhargava. *Dealing with Mobility: Issues and Research Challenges*. Technical Report, Purdue University (1993).
- [26] C. Pu & A. Leff. Autonomous Transaction Execution with Epsilon Serializability. In *Proceedings of 1992 RIDE Workshop on Transactions and Query Processing*, pp. 2-11 (1992).
- [27] K. Ramamritham. Real-time Databases. *International Journal of Distributed and Parallel Databases*, 1(2):199-226 (1993).
- [28] L. Sha, R. Rajkumar, S.H. Son and C.H. Chang. A Real-time Locking Protocol. *IEEE Transactions on Computers*, 40(7):793-800 (1991).
- [29] S.H. Son, S. Park & Y. Lin. An Integrated Real-time Locking Protocol. In *Proceedings of the International Conference on Data Engineering*, pp. 527-534 (1992).
- [30] O. Ulusoy, & G.G. Belford. Real-time Transaction Scheduling in Database Systems. *Information Systems*, 18(8):559-580 (1993).
- [31] O. Ulusoy. A Study of Two Transaction Processing Architectures for Distributed Real-time Database Systems. *Journal of Systems and Software*, 31(2): 97-108 (1995).
- [32] O. Ulusoy. Real-Time Data Management for Mobile Computing. In *Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98)*, Berlin, Germany, pp. 233-240 (1998).
- [33] O. Ulusoy & A. Buchmann. A Real-Time Concurrency Control Protocol for Main-Memory Database Systems. *Information Systems*, 23(2):109-125 (1998).
- [34] P.S. Yu, K.L. Wu, K.J. Lin and S.H.Son. On Real-Time Databases: Concurrency Control and Scheduling. In *Proceedings of IEEE*, 82(1):140-57 (1994).
- [35] P. Xuan, O. Gonzalez, J. Fernandez and K. Ramamritham. Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments. *Proceedings of 3<sup>rd</sup> IEEE Real-Time Technology Application Symposium*, pp. 38-48 (1997).
- [36] M. Xiong, K. Ramamritham, R. Sivasankaran, J.A. Stankovic, and D. Towsley. Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics. In *Proceedings of IEEE Real-Time Systems Symposium*, pp. 240-251 (1996).

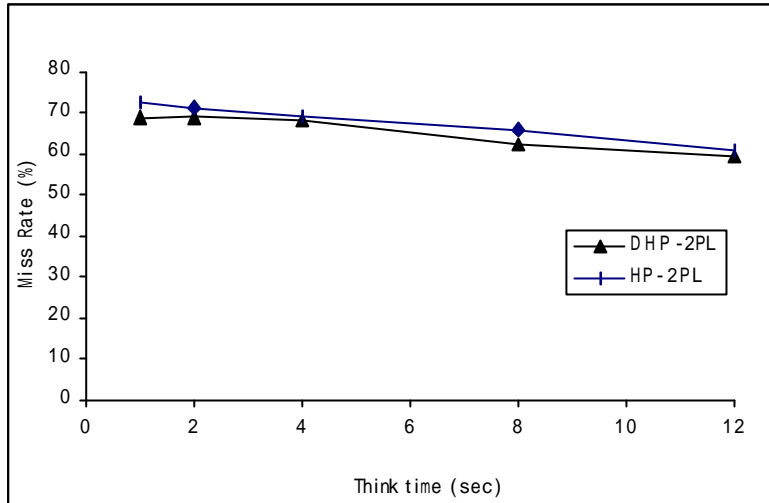


Fig. 6: Impact of Think Time on Miss Rate

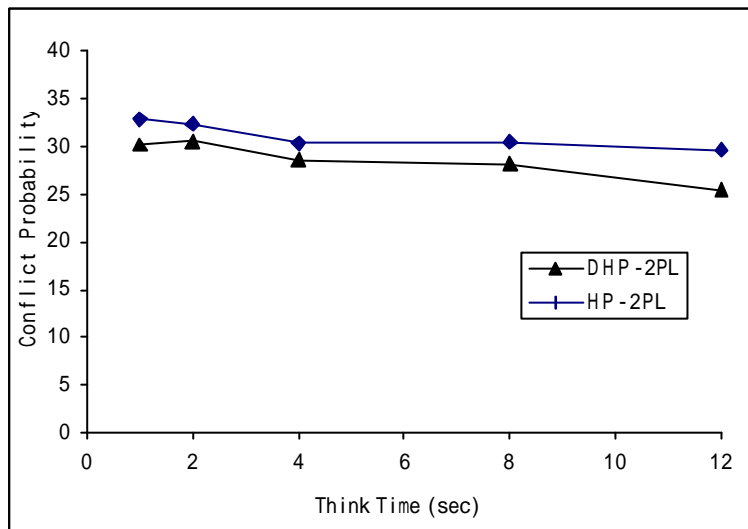


Fig. 7: Impact of Think Time on Conflict Probability

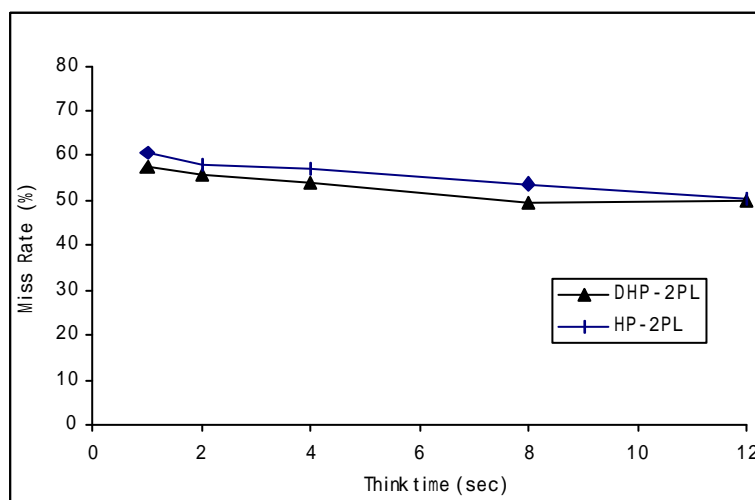


Fig. 8: Impact of Think Time on Miss Rate when the slack bound is 25-20

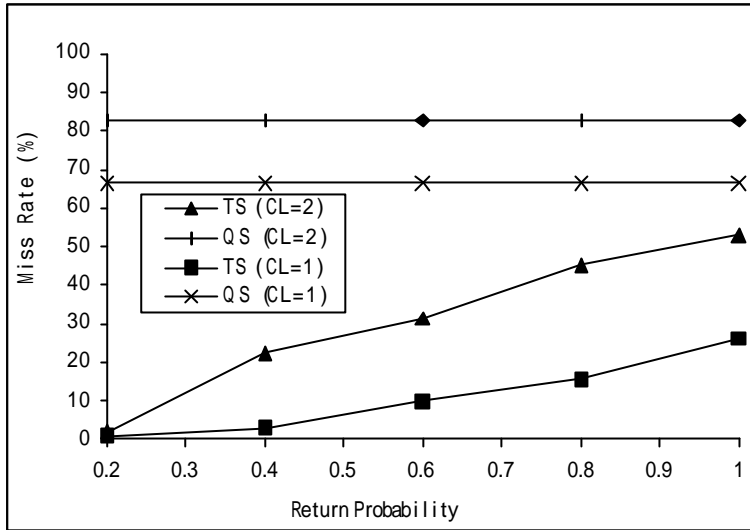


Fig. 9: Impact of Return Probability on Miss Rate

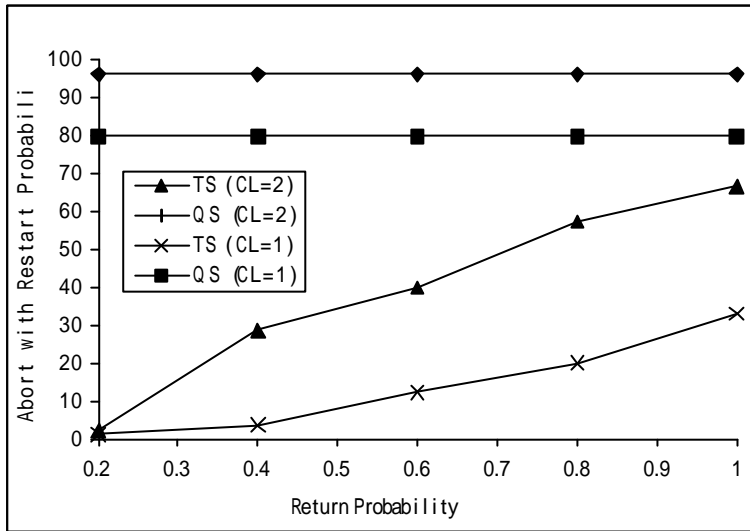


Fig. 10: Impact of Return Probability on Abort with Restart Probability

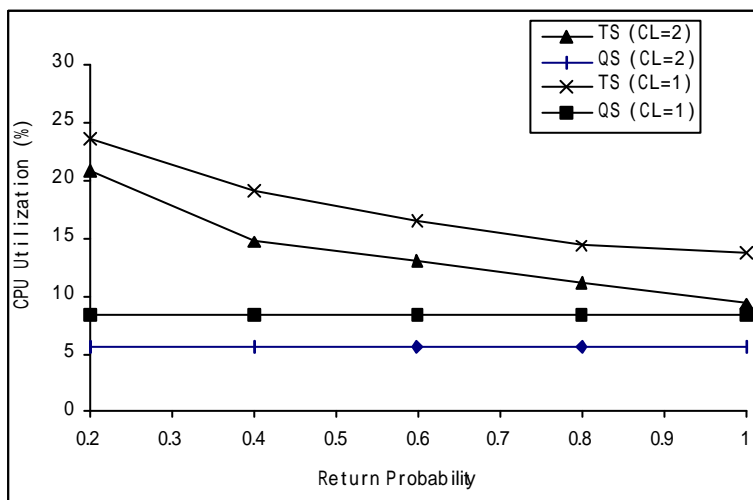


Fig. 11: Impact of Return Probability on CPU Utilization

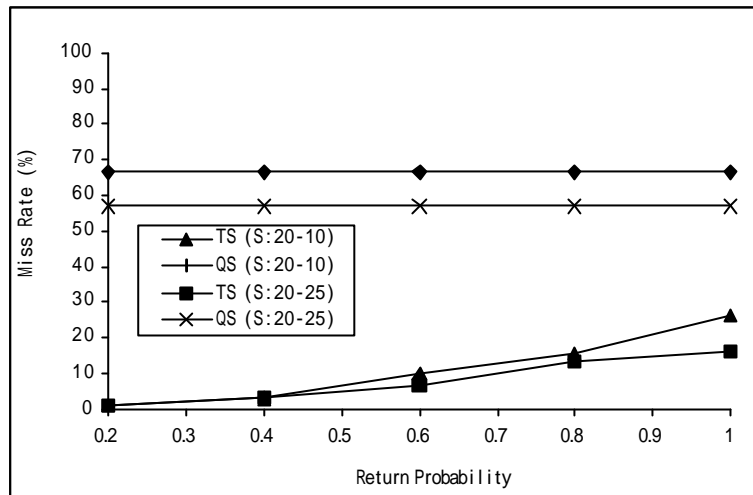


Fig. 12: Impact of Return Probability on Miss Rate

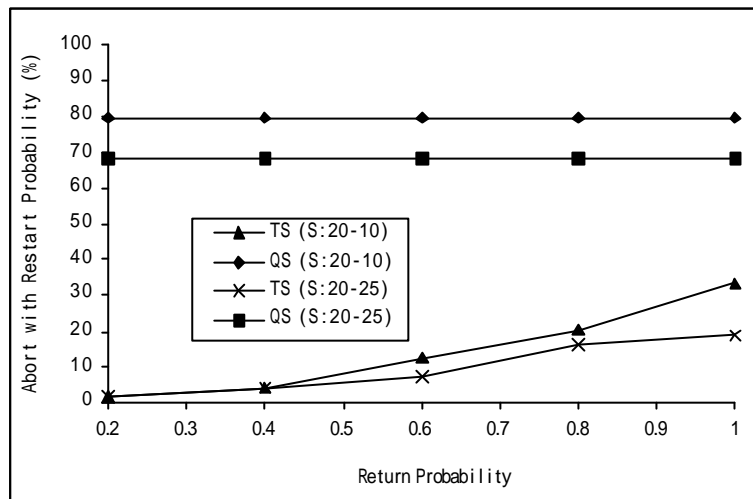


Fig. 13: Impact of Return Probability on Abort with Restart Probability

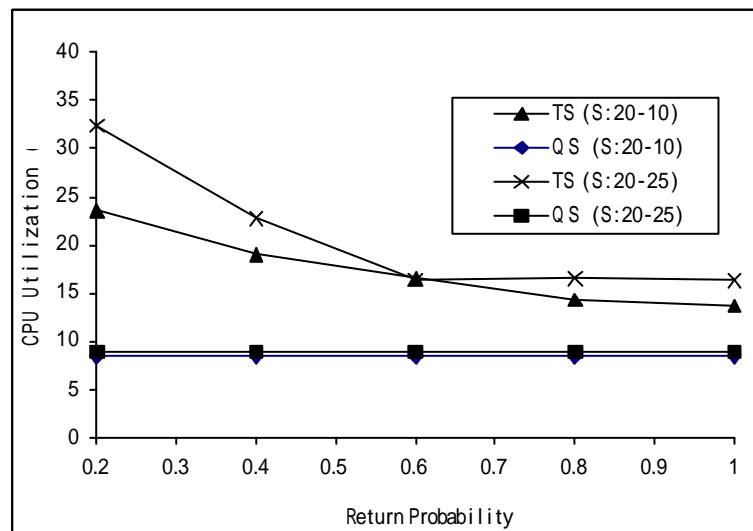


Fig. 14: Impact of Return Probability on CPU Utilization

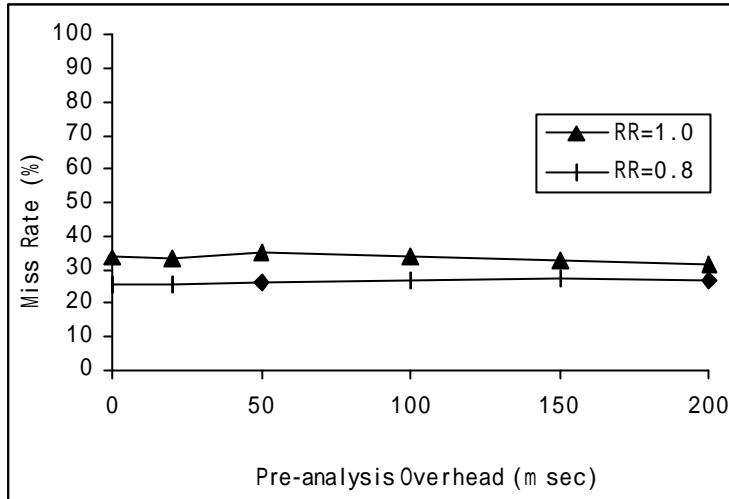


Fig. 15: Impact of Pre-analysis Overhead

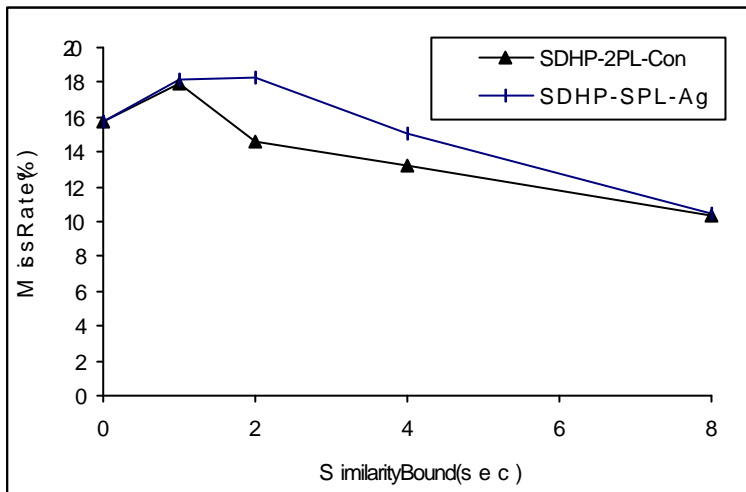


Fig. 16: Impact of Similarity Bound on Miss Rate

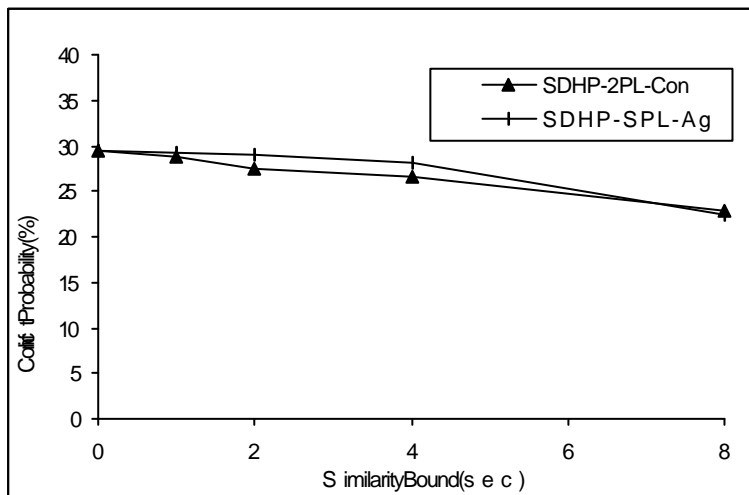


Fig. 17: Impact of Similarity Bound on Conflict Probability

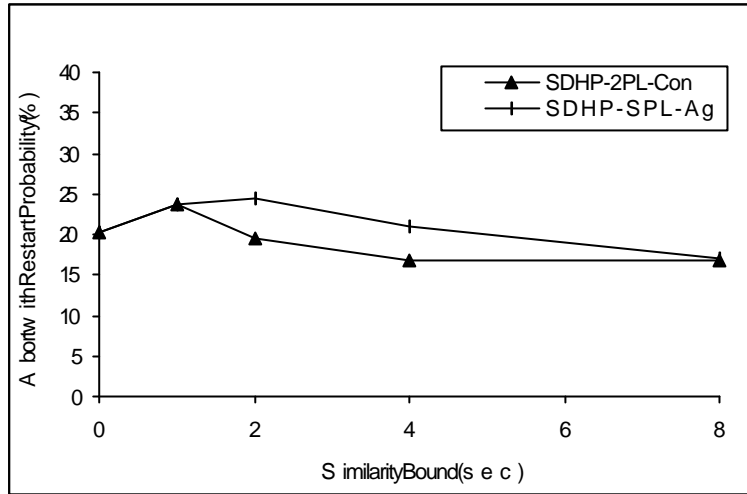


Fig. 18: Impact of Similarity Bound on Abort with Restart Probability