

A Similarity-Based Protocol for Concurrency Control in Mobile Distributed Real-Time Database Systems*

Kam-yiu Lam¹, Tei-Wei Kuo², Gary C.K. Law¹ and Wai-Hung Tsang¹

Department of Computer Science¹
City University of Hong Kong
83 Tat Chee Avenue, Kowloon
HONG KONG
Email: cskylam@cityu.edu.hk

Department of Computer Science and
Information Engineering²
National Chung Cheng University
Chiayi, 621 Taiwan, ROC
email: ktw@cs.ccu.edu.tw

Abstract

Research in the concurrency control of real-time data access over mobile networks is receiving growing attention. With possibly lengthy transmission delay and frequent disconnection, traditional concurrency control mechanisms may become very costly and time-consuming in mobile distributed real-time database systems (MDRTDBS). Due to limited bandwidth and unpredictable behavior of mobile networks, the past research on concurrency control for distributed real-time database systems (DRTDBS) can not be directly applied to MDRTDBS. In this paper, we propose a distributed real-time locking protocol, called SDHP-2PL, based on the High Priority Two Phase Locking (HP-2PL) scheme and the concept of similarity for MDRTDBS. We consider the characteristics of mobile networks and adopt the concept of similarity to reduce the possibility of lock conflicts. A detailed model of a MDRTDBS has been developed, and a series of simulation experiments have been conducted to evaluate the capability of SDHP-2PL and the effectiveness of using similarity for MDRTDBS. The simulation results have confirmed our belief that the use of similarity as the correctness criterion for concurrency control in MDRTDBS can significantly improve the system performance.

1 Introduction

In recent years, the research in *mobile distributed real-time database systems* (MDRTDBS) is receiving growing attention due to a large number of potential mobile computing applications, such as real-time traffic monitoring systems and mobile internet stock trading systems [3]. 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, which are quite different from those developed for *distributed real-time database systems* (DRTDBS) over wired networks [7,9,10]. How to meet the timing constraints of transactions over a mobile network and, at the same time, to maintain the external and temporal consistency of the databases involves various challenging research issues [10]. Up to now, to our knowledge, it is still lack of a detailed study on the design of MDRTDBS.

Concurrency control has been an active research topic for *real-time database systems* (RTDBS) [6,11]. In the last decade, researchers have proposed various real-time concurrency control protocols, e.g., [1,5], for single-site as well as for DRTDBS. Efficient techniques were proposed to bound the number of priority inversions [1]. In RTDBS, priority inversion is, in general, resolved by either transaction restarts or by priority inheritance [2]. For example, the High Priority Two Phase Locking (HP-2PL) protocol [1] restarts a lower-priority transaction if there exists a higher-priority transaction that wants to set a lock on a data object currently

* Supported in part by a research grant from the National Science Council under Grants NSC88-2213-E-194-034 and Strategic Grant # 7000584 City University of Hong Kong.

locked by the lower-priority transaction in a conflicting mode.

Although many excellent concurrency control protocols have been proposed for RTDBS and DRTDBS, the proposed protocols, in general, can not be directly applied to MDRTDBS due to the unique characteristics of MDRTDBS [7,10]. In a MDRTDBS, whether a transaction can meet its deadline highly depends on the properties of the communication network. The unpredictable propagation delay of mobile networks imposes a serious time burden on the performance of MDRTDBS, and the delay can greatly affect the performance of any adopted concurrency control protocol. Compared with wired networks, a mobile network is often much slower, unreliable, and unpredictable. The mobility of clients in a mobile computing system also greatly affects the distribution of workload in the communication network. Disconnection between clients and stations is common [3,7]. The poor quality of service provided by a mobile network usually seriously increases the overheads in resolving data conflicts and affects the performance of existing concurrency control protocols which do not consider the characteristics of mobile networks.

This paper proposes a distributed real-time locking protocol, called SDHP-2PL which is based on the High Priority Two Phase Locking (HP-2PL) scheme [1] and the concept of *similarity* [4,5], for MDRTDBS. We consider the characteristics of mobile networks and adopt the concept of *similarity* to reduce the possibility of lock conflicts and to increase the system concurrency, which is vital to systems “virtually divided” by slow and unreliable mobile networks. In the SDHP-2PL, the priority inheritance mechanism is mixed with semantics-based concurrency control mechanisms to boost the performance of the proposed locking protocol for data access over a mobile network.

2 Model of a Mobile Distributed Real-Time Database System

2.1 System Architecture

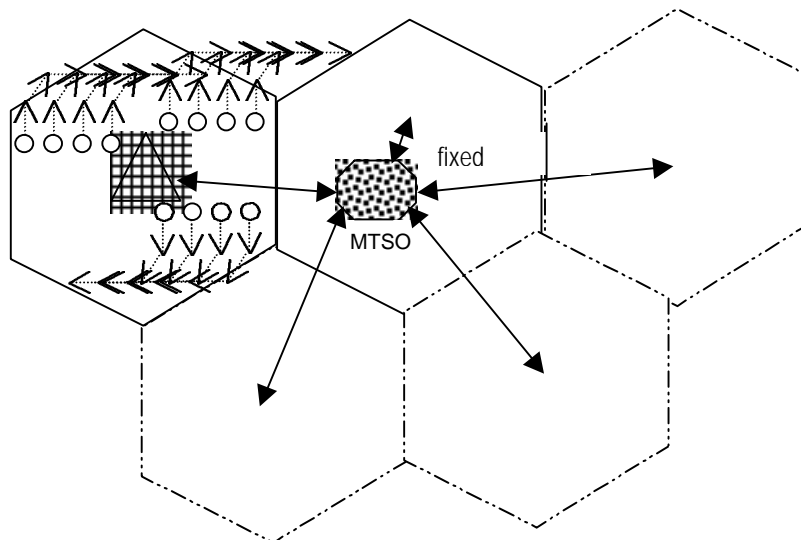


Figure 1: System Architecture of the Mobile Distributed Real-time Database System

A MDRTDBS consists of three major components: mobile clients (MC's), base stations, and a main terminal switching office (MTSO), as shown in Figure 1. The entire service area is divided into a number of connected cells. Within each cell, there is a base station that is augmented with a wireless interface to communicate with the MC's within its cell. The base stations in different cells are connected to the MTSO by a point-to-point wired network.

Attached to each base station is a database system containing a local database which may be accessed by transactions issued by MC's within its cell or from the other cells via the MTSO. The MTSO is responsible for active call information maintenance, handoff procedure, channel allocation, and message routing.

An MC may move around within a cell or cross the cell border to enter another cell. Periodically, an MC sends a location signal to its assigned base station. The strength of the location signal received by the base station depends on the distance between the MC and the base station. When an MC is crossing the cell border, the signal may become very weak. When the signal received by the base station is lower than a certain threshold level, the MTSO will be notified. The MTSO will perform a handoff procedure. It sends out requests to all the base stations to acquire their responses regarding the strength of the location signal received by them from the MC. The MTSO will then re-assign the MC to the base station which has received the strongest signal from the MC. Usually this is the base station which is responsible for the cell which the MC is entering.

When a transaction is generated from an MC, the MC will first attempt to issue a call request to the base station of its cell. A channel is granted to the MC after the completion of a setup procedure. The execution of the setup procedure incurs a fixed overhead as it involves communication amongst the MC, the base station, and the MTSO. Since the number of channels for communication between a base station and its MC's is limited, it is possible that the channel request is refused due to no free channel available. If the number of attempts issued by the MC exceeds a certain specified threshold number, the call and the transaction will be aborted. Due to slow and unreliable communication, and channel contention, the time required to establish a channel is highly unpredictable. Once a channel is established, the transaction will be sent out through the RF transmitter (which is connected to the antenna) from the MC to the base station of the cell where the MC currently resides. When an MC is crossing the cell border while it is communicating with its base station, a new channel will be created with its newly assigned base station after a setup procedure is done by the MTSO.

2.2 Database and Transaction Models

The database is distributed among the base stations. At each base station, there is a local database that consists of two types of data objects: *temporal* and *non-temporal* data objects. Temporal data objects are used to record the status of external objects in the real world. Each temporal data object is associated with a timestamp that denotes the age of the data object. A transaction is given a timestamp, when the transaction is initiated, if the transaction may update a temporal data object. If the transaction commits successfully before its deadline, the timestamp of the data object updated by the transaction will be set as the timestamp of the transaction. The validity of a temporal data object is indicated by an *absolute validity interval (avi)* [8]. A temporal data object satisfies the *avi* constraint if the age of the data object is up to date, i.e., the difference of the current time and the age is no more than *avi*. A *relative validity interval (rvi)* may also be given to a transaction which requires that the maximum age difference of data objects read by the transaction is no larger than *rvi* [8].

An MC may initiate a transaction sporadically. Transactions from the MC's are consisting of a sequence of read and write operations. Each transaction issued from an MC is associated with a deadline and a numerical value expressing its criticality. The priority of a transaction is derived from these values. The operations may access data objects residing at several base stations. When an operation of a transaction accesses data objects residing on a base station other than the MC's current one, the operation will be routed to the base station via the MTSO. When all the operations of a transaction have been processed, a commit protocol will be performed to ensure the failure atomicity of the transaction.

3 Similarity-Based Concurrency Control Protocol

3.1 The Basic Strategy

3.1.1 Data Similarity

Similarity is closely related to the important idea of imprecise computation in real-time systems and to the idea of partial computation for database. For many real-time applications, the value of a data object that models an entity in the real world cannot, in general, be updated continuously to perfectly track the dynamics of the real-world entities. It is also *unnecessary* for data values to be perfectly up-to-date or precise to be useful. In particular, data values of a data object that are slightly different are often interchangeable. The concept of *similarity* can be used to extend the conventional correctness criteria for concurrency control in database systems [4,5] and to balance data precision (based on *similarity*) and system workload. If two operations in a schedule are strongly similar, then they can always be used interchangeably in a schedule without violating the integrity and consistency of the database.

3.1.2 The Basic Strategy for Conflict Resolution

Specifically, we assume that the application semantics allows us to derive a similarity bound for each data object such that two write operations on a data object must be similar if their time-stamps differ by an amount not greater than the similarity bound. The basic strategy for conflict resolution in the similarity-based Distributed High Priority Two Phase Locking (SDHP-2PL) can be summarized as follows:

- (i) Suppose that a transaction T_i issues a read-lock request on data object D_k , and D_k is already write-locked by a collection of transactions $TH = \{T_j, \dots, T_y\}$. 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 should be granted because T_i will always read *similar* data values, regardless of which transaction in TH commits (or updates D_k first).
- (ii) Suppose a transaction T_i issues 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 should be granted. It is because the final database state will be similar regardless of which write operation is performed first.
 - (2) If TR is not empty, and the conflicting write operation of T_i , the conflicting write operation of T_j (the last committed transaction on D_k), and the conflicting write operations of all transactions in TW are similar, then the write-lock of T_i should be granted. It is because the final database state will always be *similar* and the read operations will read *similar* data values regardless of which write operation is performed first.

Obviously, if the above *similarity* test of a lock request fails, the lock-requesting transaction should be blocked or the transactions which cause the blocking might be restarted. In the SDHP-2PL, we adopt the following policies:

Suppose that the *similarity* test fails for the lock request of transaction T_i on data object D_k , and TH is a set of transactions which currently read-lock or write-lock D_k . 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 a priority lower than T_i . If the restart of all transactions in SCT will make T_i passing the *similarity* test, then the transactions in SCT are restarted, and the lock request of T_i is granted. Otherwise, T_i is blocked. Thus, blocking will be used if the priority of any of the lock-holding transactions is higher than the priority of the lock requesting transaction.

3.2 The Protocol

In SDHP-2PL, it is assumed that a distributed lock scheduler is deployed at each base station. The lock scheduler of a base station manages the lock requests for the data items T_r request a lock in a specified mode, i.e., read or write, on a data object O_k residing at a base station. T_r should invoke the lock request procedure of SDHP-2PL, which is defined as follows, to receive the lock. The invocation may cause the restart of lower-priority transactions, the blocking of T_r , or the granting of the lock request.

```

Lock-Request ( $T_r$ , Mode,  $O_k$ )
Begin
  If the lock request of  $T_r$  is similar to existing locks on
  data object  $O_k$ 
  Then
    /* Please see Section 3.1.2 for the similarity of locks */
    Return(Success)
  Else
    Let CT be the set of transactions that lock  $O_k$  in a mode
    conflicting and non-similar to the lock request of  $T_r$ 
    For each transaction  $T_c$  in CT Do
      If Priority( $T_r$ ) > Priority( $T_c$ )
      Then
        If  $T_c$  is not committing and restarting
        Then
          Restart  $T_c$  locally (or globally) if  $T_c$  is a
          local (or global) transaction.
          Remove  $T_c$  from CT
        Else
          Priority( $T_c$ ) = Priority( $T_r$ ) + C
        EndIf
      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)
  EndIf
End

```

If the lock request is *similar* to the existing locks on the same data object, then the lock request is granted. Otherwise, the set of transactions, which lock O_k in a mode conflicting and *non-similar* to the lock request of T_r , is considered for restart or priority inheritance. If a transaction T_c in CT has a lower priority than the priority of T_r , and the transaction is committing or restarting, then we should accelerate the execution of T_c to release the lock that blocks T_r . The acceleration is done by letting T_c inherit the priority of T_r , where C is a tunable constant to further boost the priority of transactions that blocks T_r , such that T_c can complete as soon as possible.

If a transaction T_c in CT has a priority lower than T_r , and T_r is not committing or restarting, then we simply restart T_c . If T_c is a local (or global) transaction, we restart it locally (or globally). When a transaction is restarted, no lock-requesting transactions can get any lock on any data object locked by the restarted transaction until the undo operations of the restarted transaction are completed. The time required to complete the undo operations can be shortened by using of the idea of priority inheritance.

Note that the SDHP-2PL is a variation of HP-2PL, and HP-2PL is deadlock-free. Obviously, the priority inheritance mechanism in the lock request procedure of SDHP-2PL will

not introduce any type of deadlocks to HP-2PL because the priority inheritance mechanism is only used to resolve lock conflicts between T_r and committing transactions. When there exist transactions which lock some data object O in a mode conflicting and non-similar to the lock T_r

Furthermore, the *similarity* request to be granted, and it surely will not introduce any deadlock to the system. Therefore, SDHP-2PL is deadlock-free.

4 Performance Studies

4.1 Simulation Model

We have built a detailed simulation program for the MDRTDBS model discussed in Section 2. The simulation program implements the essential functionality of a mobile personal communication system (PCS) and a DRTDBS. For instance, the call setup and tear down features are modeled explicitly. The mobile clients move around and may cross the cell border into another cell. Based on the received signal strength from each base station, they can self-locate themselves. The handoff procedure is handled by the MTSO. In the system level, we have implemented a DRTDBS.

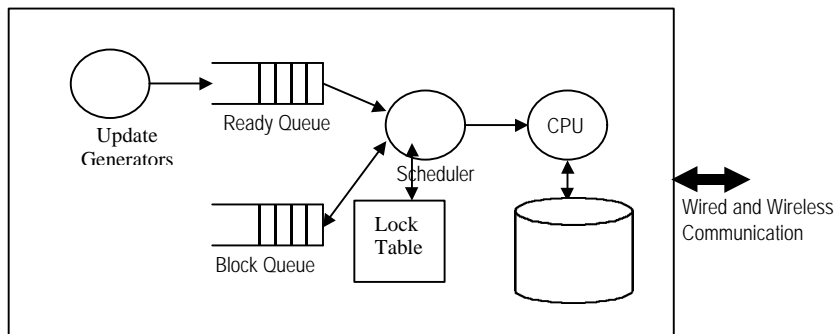


Figure 2: Model of the database system in a base station

The database system at each base station consists of a scheduler, a CPU with a ready queue, a local database, a lock table, and a block queue, as shown in Figure 2. Since a main-memory database system usually better supports real-time applications [1], it is assumed that the database resides in the main memory to eliminate the impact of disk scheduling on the system performance. In order to focus our simulation objectives and exclude unnecessary factors, it is assumed that all the temporal data objects have the same *avi* and *rvi*. 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 the transactions in order to maintain the validity of the temporal data objects [8].

Transactions are generated from the mobile clients sporadically. It is assumed that all the transactions are firm real-time transactions. Each transaction is defined as a sequence of operations. Once a transaction is generated at a mobile client, it will be forwarded to that base station. A coordinator process will be created at the base station. It is responsible for the processing and committing of the transaction. The operations in a transaction will be processed one after one. If the required data item of an operation is located at another base station, the transaction will be forwarded to the base station. Occasionally, an operation may need to return to its originating mobile client for getting input data. We define the probability of an operation returning to the originating mobile client as the *return probability* (RR). Return probabilities can greatly affect the performance of the system as a higher return probability means a greater dependency on the mobile network.

Transactions wait in the ready queue for CPU allocation, and the CPU is scheduled according to the priorities of the transactions. The scheduler will check the deadline before a transaction is allocated the CPU. If the deadline is missed, the transaction will be aborted immediately. If any of the temporal data objects accessed by the transaction is invalid before the commit of the transaction, the transaction will also be aborted as it has read some out-dated data objects. After the completion of a transaction, the mobile client will generate another transaction after a think time.

4.2 Model Parameters and Performance Measures

Similar to many previous studies on RTDBS and DRTDBS, the deadline of a transaction, T , is defined according to the expected execution time of a transaction [1,9]. The following table lists the model parameters and their base values.

Parameters	Baseline Values
Number of MTSO	1
Number of Cells	7
Location Update Interval	0.2 second
Transmission Speed for Channel	10 kbps
Number of Channels for each Cell	10
Think Time	8 seconds
Transaction Size	7 to 14 operations, uniform distribution
Write operation probability (WOP)	1
Return Probability	0.6
Slack Factor	Between 8 and 12 (uniformly distributed)
Similarity bound (SB)	15 seconds
Number of Mobile Clients	84
Channel Connection time (T_{comm})	1 second
Call Update Interval	0.2 second
Number of Local Databases	7 (1 in each base station)
Database Size	100 data objects per local database
Fraction of Temporal Data Objects	10%
Temporal Data Object Update Interval	0.5 update per second
Absolute Threshold	12 seconds
Relative Threshold	8 seconds

The channel connection time is the time required to send a message from an MC to its base station. It is defined 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 is 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.

The primary performance measure used is the miss rate (MR) which is defined as the number of deadline-missing transactions over the total number of transactions generated in the system. In addition to the miss rate, we also measure the restart probability which is defined as the total number of restarts over the number of committed transactions.

4.3 Simulation Results and Discussions

In the experiments, we test the performance of the SDHP-2PL under different similarity bounds to investigate the effectiveness of using the concept of similarity as the correctness criterion for concurrency control. When the similarity bound is set to zero, the

protocol is similar to a distributed HP-2PL except that priority inheritance is used to resolve the conflict when the lock holder is committing.

Figure 3 gives the impact of different similarity bounds on MR at return probabilities of 0.6 and 0.8. As explained in section 4.1, the return probability defines the probability of a transaction returning to its originating mobile client after the completion of an operation. An increase in return probability increases the workload on the mobile network as a transaction will have a higher probability to go back to its originating mobile client. Thus, the miss rate is higher at a higher return probability. In Figure 3, we can see that although the improvement in MR is marginal if the similarity bound is small, a great improvement is achieved for large similarity bounds. The miss rate at similarity bound of 30 sec is only about 40% of that at the similarity bound of 5 sec. The improvement is due to smaller number of blocking and restart probability. When the similarity bound is large enough, e.g., 15 sec or greater, more transactions may be allowed to access the locks in conflicting modes in the SDHP-2PL if the conditions for the similarity test (defined in section 3.2) are satisfied. However, if the similarity bound is zero or small, more transaction restarts, and blocking will occur. The probability of deadline missing will be much higher. The restarted transactions increase the system workload, especially the communication channels between the mobile clients and the base stations. Thus, the channel blocking probability will be higher and the transactions will require a much longer time for completion. The small improvement in MR at small similarity bounds may come from the fact that even though a transaction may be allowed to set a lock (if it is similar to all the existing lock holders), other transactions may not be similar. Thus, blocking and restarts may still occur. Furthermore, the number of transaction restart will be greater and the blocking time will be longer due to a greater number of lock holders.

The smaller probability of transaction restart due to similarity can be observed in Figure 4. One interesting fact we can see from Figure 4 is that the restart probability at return probability of 0.6 is higher than that at return probability of 0.8 except at large similarity bounds, e.g., greater than 15 sec. The reason is that at a higher return probability, more transactions will be aborted before they set any locks due to deadline missing.

The impact of return probability on the performance of the system at a similarity bound of 15 sec is shown in Figure 5 and Figure 6. It can be seen that the use of *similarity* can consistently improve the system performance at different return probabilities. Greater improvement is obtained at smaller return probabilities. The reason is that at a larger return probability, the transactions will have a higher probability of deadline missing due to fewer channel contention (even though they do not have any data conflict). This can be observed in Figure 6 in which we can see that the restart probability is higher for similarity bound of 0 second than a similarity bound of 15 seconds.

Figure 7 depicts the miss rate at different write operation probabilities of transactions. It can be observed that the improvement will be greater for the write operation probabilities between 0.3 to 0.5. If the operations are mainly write operations, the conditions is more difficult to be satisfied as the conditions is more restrictive for write operations. (Please refer to section 3 for the details.) If most of the operations are read operations, the probability of lock conflict will be low. At a write operation probability of zero, there is no lock conflict as all the lock requests are read locks. Thus, the performance of the system at similarity bound of 15 seconds is the same as at similarity bound of 0 second. Increasing the write operation probability increases the lock conflict probability. Thus, the difference between similarity bound = 15 and similarity bound = 0 becomes greater. When there are a large number of write operations, the probability of satisfying the conditions in *similarity* check will be smaller. Thus, the improvement becomes smaller.

5 Conclusions

This paper proposes a real-time locking protocol, called SDHP-2PL, based on the High Priority Two Phase Locking (HP-2PL) scheme [1] and the concept of *similarity* for

MDRTDBS. We consider the characteristics of mobile networks and adopt the concept of *similarity* to reduce the possibility of lock conflicts and to increase the concurrency degree of the system, which is vital to systems “virtually divided” by slow and unreliable mobile networks. The priority inheritance mechanism is mixed with semantics-based concurrency control mechanisms to boost the performance of the proposed locking protocol for data access over a mobile network. We propose a detailed model for MDRTDBS and conduct a series of simulation experiments to evaluate the capability of SDHP-2PL and the effectiveness of using *similarity* for MDRTDBS. Since bandwidth is one of the most scarce resources in a mobile environment, the study shows that the use of *similarity* can significantly improve the system performance by reducing the number of lock conflicts. We also observe that the effectiveness of *similarity* depends highly on the return probability of the transactions and the read operation probability. The use of *similarity* can significantly improve the system performance in terms of miss rate. This is especially true when the probability of write operations is higher and the return probability is not very high.

References

- [1] R. Abbott and Hector Garcia-Molina, “Scheduling Real-Time Transactions: A Performance Evaluation,” *ACM Trans. on Database Systems*, vol. 17, no. 3, pp. 513-560, Sep. 1992.
- [2] J. Huang, J. Stankovic & K. Ramamritham, “Priority Inheritance in Soft Real-Time Databases”, *Journal of Real-Time Systems*, vol. 4, no. 3, pp. 243-268, 1992.
- [3] T. Imielinski and B. R. Badrinath, “Mobile Wireless Computing: Challenges in Data Management,” *Communications of the ACM*, vol. 37, no. 10, Oct. 1994.
- [4] T.W. Kuo and A.K. Mok, “Application Semantics and Concurrency Control of Real-Time Data-Intensive Applications,” *Proceedings of IEEE 13th Real-Time Systems Symposium*, 1992.
- [5] T.W. Kuo and A.K. Mok, “SSP: a Semantics-Based Protocol for Real-Time Data Access,” *Proceedings of IEEE 14th Real-Time Systems Symposium*, December 1993.
- [6] G. Ozsoyoglu and R. Snodgrass, “Temporal and Real-Time Databases: A Survey”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 4, pp. 513-532, 1995.
- [7] E. Pitoura and Bharat Bhargava, “Revising Transaction Concepts for Mobile Computing,” *Proc. Workshop on Mobile Computing Systems and Applications*, pp. 164-168, USA, 1995.
- [8] K. Ramamritham, “Real-time Databases”, *International Journal of Distributed and Parallel Databases*, vol. 1, no. 2, 1993.
- [9] O. Ulusoy, “Processing of Real-time Transactions in a Replicated Database Systems”, *Journal of Distributed and Parallel Databases*, vol. 2, no. 4, pp. 405-436 1994.
- [10] O. Ulusoy, “Real-Time Data Management for Mobile Computing”, *Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98)*, Berlin, Germany, July 1998.
- [11] P.S. Yu, K.L. Wu, K.J. Lin & S.H.Son, “On Real-Time Databases: Concurrency Control and Scheduling,” *Proceedings of IEEE*, vol. 82, no. 1, pp. 140-57, 1994.

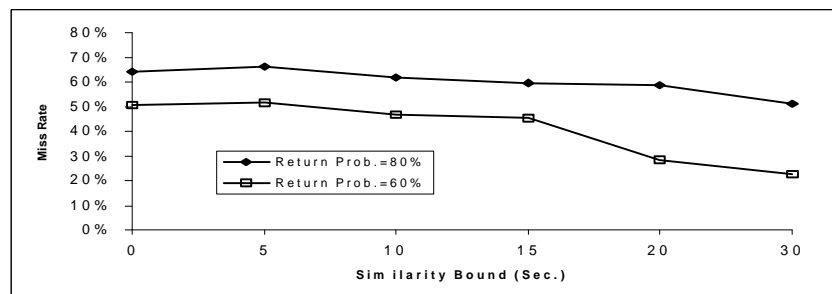


Figure 3. Miss Rate at Different Similarity Bounds (WOP = 1)

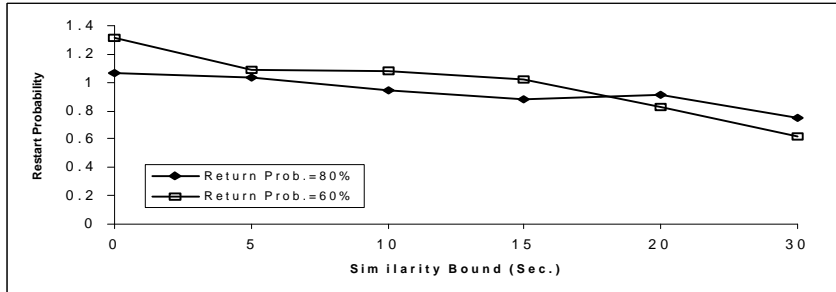


Figure 4. Restart Probability at Different Similarity Bounds (WOP = 1)

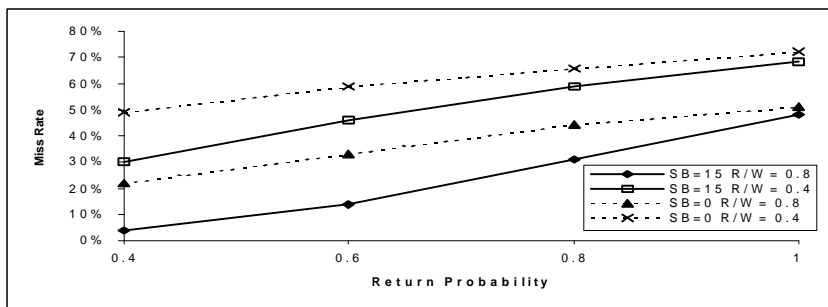


Figure 5. Miss Rate at Different Return Probability

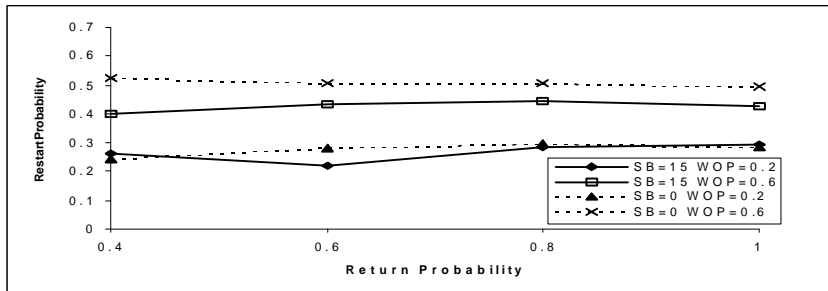


Figure 6. Restart Probability at Different Return Probability

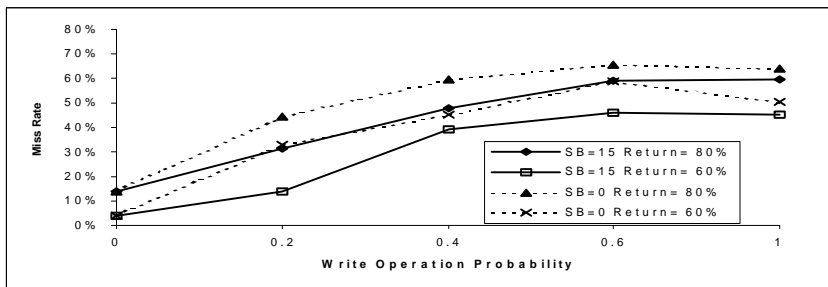


Figure 7. Miss Rate at Different Write Operation Probability