A Secure Real-Time Concurrency Control Protocol for Mobile Distributed Real-Time Databases

***Yingyuan Xiao*[†], *Yunsheng Liu*[††], *Guoqiong Liao,* [†††],**

[†]Department of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300191, P.R. China
[††]School of Computer Science and Techn., Huazhong University of Science and Technology, Wuhan 430074, P.R. China
[†††]China Corporate Technology, Siemens Limited, Beijing 100102, P.R. China

Summary

A class of security-critical applications with the requirements of timing constraints, such as wireless stock trading, power network scheduling, real-time traffic information management, etc., demand the support of mobile distributed real-time database systems. For the class of applications, mobile distributed real-time database systems must simultaneously satisfy two requirements in guaranteeing data security and minimizing the missing deadlines ratio of transactions. Multilevel secure database systems based on mandatory access control can prevent direct unlawful information flows between transactions. However, they can't be free from illegal information transfers between transactions belonging to different clearance levels by means of the concurrency control mechanism. This paper presents a secure hybrid optimistic real-time concurrency control protocol (SHORTCCP) for mobile distributed real-time database systems. The SHORTCCP not only fully considers the intrinsic limitations of mobile environments and the requirements of timing constraints of real-time transactions, but also makes an effective tradeoff between security and real-time performance by introducing secure influence factor and real-time influence factor. The experimental results show the SHORTCCP achieves data security without degrading real-time performance significantly.

Key words:
Mobile distributed real-time databases; Concurrency control; Security; Covert channel; Deadline
.

1. Introduction

From A class of security-critical applications with the requirements of timing constraints, such as wireless stock trading, power network scheduling, real-time traffic information management, etc., demand the support of mobile distributed real-time database systems. A mobile distributed real-time database system (MDRTDBS) is, in general, defined as a distributed real-time database system supported by a mobile environment. For MDRTDBSs, transactions and data can have timing characteristics or explicit timing constraints. The timing constraints of transactions are typically specified in the form of deadlines that require a transaction to be completed by a specified time. For soft real-time transactions, failure to meet a deadline can cause the results to lose their value, and for firm or hard real-time transactions, a result produced too late may be useless or harmful, so concurrency control and schedule strategies suitable for MDRTDBSs must priorly ensure the transactions with urgent deadlines. In a MDRTDBS, transactions are given priorities that are used when scheduling transactions and resolving data conflicts, and the priority assigned to a transaction is directly related to the deadline of the transaction. For instance, transactions are assigned priorities that are directly proportional to their deadlines in Earliest Deadline First (EDF) [1]. That is, the transaction with the nearest deadline gets the highest priority. Owing to the intrinsic limitations of mobile environments, such as mobility, frequent disconnection and limited bandwidth, it will became more difficult to meet transaction deadlines in a mobile environment.

MDRTDBSs are usually applied to the safety-critical applications. For these applications, besides the demand for meeting transaction deadlines, it is essential to prevent unlawful information flows between different transactions. The traditional real-time concurrency control protocols for single-site as well as (mobile) distributed real-time database systems, e.g., [2, 3, 4, 5, 6, 7, 8, 9,10, 11] mainly focus on minimizing the ratio of transactions missing their deadlines, and don't consider the possible covert communications between transactions by means of the concurrency control mechanism. On the other hand, most secure database systems don't take account of the timing constraints of transactions and data, so can't guarantee the real-time performance. As a result, the researches on the secure real-time concurrency control protocols suitable for

Manuscript received January 5, 2007
Manuscript revised January 25, 2007

MDRTDBSs, which achieve data security without sacrificing real-time performance significantly, have very important significations.

Some secure real-time concurrency control protocols for single site real-time database systems have been presented [112, 13, 14, 15, 16, 17, 18, 19], but these protocols hardly consider the intrinsic limitations of mobile distributed environments, and can't make a suitable tradeoff between data security and real-time requirements (usually emphasizing one hand and softening up the other hand).

This paper presents a secure hybrid optimistic real-time concurrency protocol (SHORTCC) for MDRTDBSs. The SHORTCC combines optimistic concurrency control with two-Phase Locking High Priority (2PLHP) in mobile distributed real-time transaction processing, and makes a suitable tradeoff between guaranteeing data security and minimizing the missing deadlines ratio of transactions by introducing secure influence factor and real-time influence factor.

2. Concurrency Control Covert Channel

Most secure database systems have access control mechanisms based on the Bell-LaPadula model, which is specified in terms of subjects and objects [20]. An object is a data item, whereas a subject is an active process, such as a transaction, which requests access to an object. Each object in the system has a classification level based the security requirement. Similarly, each subject has a corresponding clearance level based on the degree to which it is trusted by the system. The Bell-LaPadula access restrictions ("read below" and "write above") can prevent direct unlawful information flows between transactions belonging to different clearance levels. However, it is not sufficient to prevent indirect unlawful information flows, called as **concurrency control covert channel** by us, in which transactions can conspire for an illegal inter-level information transfer by means of the concurrency control mechanism. For example, if a low clearance level transaction T_1 requests access to a data item exclusively, which has already been held by a high clearance level transaction $T_{2,}$ T_1 will be delay. The presence or absence of the delay can be used to encode information by T_2 that is conspiring to pass on information to T_1.

A secure real-time concurrency control protocol must be able to avoid the concurrency control covert channel. However, the existing real-time concurrency control protocols such as 2PLHP [1], Priority Ceiling (PC) [2], OPT-SACRIFICE [3], etc., can't avoid the concurrency control covert channel. Only when a low clearance level transaction is blocked (delayed) by high clearance level transactions, can the concurrency control covert channel be caused, so the concurrency control covert channels caused by data conflicts of concurrent transactions can be prevented by improving the existing real-time concurrency control protocols according to the following strategy: when inter-level data conflict, which occurs between transactions belonging to different clearance levels, happens, the low clearance level transaction is given preference in the conflict resolution. In terms of the above strategy, low clearance level transactions can't know the presence of high clearance level transactions, i.e., no concurrency control covert channel. However, this may influence the real-time performance and result in increasing the deadline miss ratio. For example, when a low clearance level transaction which has a low priority applies a data item held by a high clearance level transaction which has a high priority, according to the improved strategy the high clearance level transaction will be aborted to let the low clearance level transaction get the data resource. Thus, the high priority transaction is aborted by the low priority transaction, i.e. priority inversion is caused. Obviously, the improved strategy guarantees data security at the cost of sacrificing the real-time performance, and can't simultaneously meet two requirements in guaranteeing data security and minimizing the missing deadlines ratio of transactions.

3. Concurrency Control Covert Channel

A typical MDRTDBS consists of the mobile hosts (MHs), the fixed hosts (FHs), the location server (LS), the mobile support base stations (MSSs), high speed fixed network and mobile network. The FHs, LS and MSSs are connected by high speed fixed network. Each FH and MSS has a database server that manages the relevant database. The LS is responsible for managing and tracking the status and current location of each MH. 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. Each MSS has a wireless communication interface and serves a cell site, which covers a definite geographical area in which MHs can communicate with the MSS. All database servers form a distributed real-time database system together to support the global mobile real-time transaction processing. Each MH transparently accesses the distributed database located at the fixed network via the MSS whose cell site covers the MH. Each database server has autonomy of site and supports the local transaction processing. In this paper, we call FH and MSS uniformly as FDS (fixed database server).

A mobile distributed real-time transaction (MDRTT) is initiated by an MH and submitted to the MSS whose cell site covers the initiating MH. Usually, there are two ways by which an MH may submit its MDRTT to the MSS: 1) the entire MDRTT is submitted in a single request message to the MSS. The MSS, which acts as the coordinator, has the control of the execution of the MDRTT. This way adapts to non-interactive MDRTT and 2) The entire MDRTT is submitted in multi-request message to the MSS. A request message submits only one operation of the MDRTT to the MSS. The MSS handles this request in the fixed network and returns the result to the MH. This way adapts to interactive MDRTT.

In our MDRTT processing model, the both ways above are supported. In the following of the paper, we mainly take the second way as an example.

An MDRTT initiated by an MH is firstly preprocessed by the MH to extract the operations ST_0 executed at the MH, to establish the operations requests of the (MDRTT – ST_0) based on the order of execution, and then orderly to submit these operations requests to the corresponding MSSs. In MDRTDBS, like conventional distributed database systems, a coordinator is required to manage the commitment of the MDRTT. Due to the mobility of an MH, the MH may suffer handoffs during the process of submitting the operations requests, so the processing of an MDRTT may involve several MSSs. An MDRTT begins with the operation of *BEGIN TID* and ends with the operation of *COMMIT* or *ABORT*, where *TID* represents the identification of the MDRTT. When an MSS receives the operation of *BEGIN TID*, it stores *TID* in its transaction queue and broadcasts the message of (*TID*, *MNA*). Here, *MNA* stands for the network address of the MSS. Other MSSs received this message store the message in their Route Table (*ROT*). The MSS received the operation of *BEGIN TID* is designated as the coordinator of the relevant MDRTT, notated by Co(MDRTT).

When the coordinator of an MDRTT receives a data operation, if possible for it to perform this operation, or dispatches the operation to some FDS (participator) based on a scheduling strategy if impossible for it to perform this operation or for the sake of optimization. Once the related FDS (coordinator or participator) receives the operation request from the coordinator, it creates a real-time subtransaction (agent) for the operation request. All the real-time subtransactions on an FDS form the local transactions set processed and controlled by the relevant database server.

Definition 1. An MDRTT is defined as the set of real-time subtransactions composing the MDRTT, i.e., MDRTT= $\{ST_i \mid ST_i$ is a real-time subtransaction composing the MDRTT, $i = 0, 1, 2, \ldots, m-1\}$, where m denotes the number of real-time subtransactions composing the MDRTT; ST_0 denotes the real-time subtransaction executed at the MH which initiates the MDRTT; ST_i ($i \neq 0$) denotes the real-time subtransactions dispatched and executed at the corresponding FDS; ST_i is defined as a 3-tuple: $ST_i ::= (O_i , C_i , <_i)$, where O_i denotes the operation set of ST_i which includes object operations, transaction operations and communication primitives; C_i denotes the timing constraints of ST_i which is usually denoted by the deadline of MDRTT; $<_i$ denotes the temporal ordering relation of O_i .

4. Secure Hybrid Optimistic Real-time Concurrency Control Protocol

4.1 Secure Influence Factor and Real-time Influence Factor

In our secure hybrid optimistic real-time concurrency control protocol (SHORTCC), secure influence factor and real-time influence factor are defined to describe respectively the severity degrees, which violate security constraints and timing constraints. In the below, we use CL(T) to denote the clearance level of the transaction T, P(T) to denote the priority of T, ST to denote the set of transaction in the system.

Definition 2. $\forall$ T$_i$, T$_j$ $\in$ ST, if exist a pair of conflict operations which belong to T$_i$ and T$_j$ respectively, T$_i$ and T$_j$ are defined as a pair of conflict transactions, notated by (T$_i$ *CF* T$_j$).

Definition 3. Suppose T$_i$ $\in$ ST, ST$_1$ $\subseteq$ ST. If the condition, $\forall$ T$_j$ $\in$ ST$_1$ (T$_i$ *CF* T$_j$), is met, ST$_1$ is said to be the Conflict Set of T$_i$, notated by CS(T$_i$).

Definition 4. Suppose T$_i$ $\in$ ST, ST$_2$ $\subseteq$ ST. If the condition, $(\forall\ T_j \in ST_2\ (T_i\ CF\ T_j\ )) \wedge (\forall\ T_j \in (\ ST- ST_2)\ (\neg\ (T_i\ CF\ T_j\ )))$, is met, ST$_2$ is defined as the Maximal Conflict Set of T$_i$, notated by MCS(T$_i$).

Definition 5. We define $|\, f\,(\mathrm{CL}(\mathrm{T}_i)) - f\,(\mathrm{CL}(\mathrm{T}_j))\,|$ as Clearance Difference Degree of between T$_i$ and T$_j$, notated by CDD(T$_i$, T$_j$). Where f is a mapping from the set of different clearance levels to the set of natural number.

Definition 6. We define $|P(\mathrm{T}_i) - P(\mathrm{T}_j)|$ as Priority Difference Degree of between T$_i$ and T$_j$, notated by PDD(T$_i$, T$_j$).

Definition 7. Suppose $T_i \in ST$, $MCS1(T_i) \subseteq MCS(T_i)$. If the condition, $(\forall\ T_j \in MCS1(T_i)\ (CL(T_j) > CL(T_i))) \wedge (\forall\ T_j \in (\ MCS(T_i) - MCS1(\mathrm{T}_i))\ (CL(T_j) \le CL(T_i)))$, is met, $MCS1(T_i)$ is defined as the High Clearance Maximal Conflict Set of T$_i$, notated by HCMCS(T$_i$).

Definition 8. Suppose $T_i \in ST$, $MCS2(T_i) \subseteq MCS(T_i)$. If the condition, $(\forall\ T_j \in MCS2(T_i)\ (P(T_j) < P(T_i))) \wedge (\forall\ T_j \in (\ MCS(T_i) - MCS2(T_i))\ (P(T_j) \ge P(T_i)))$, is met, $MCS2(T_i)$ is defined as the Low Priority Maximal Conflict Set of T$_i$, notated by LPMCS(T$_i$).

Definition 9. We define $\sum\limits_{\mathrm{Tj} \in \mathrm{HCMCS(Ti)}} (\ |\ f\,(\mathrm{CL}(\mathrm{T}_i)) - f\,(\mathrm{CL}(\mathrm{T}_j))\ |\ )$ as Secure Influence Factor of T$_i$, notated by SIF(T$_i$).

SIF(T$_i$) reflects the severity degrees of violating the security constraints caused by making T$_i$ wait in the conflict resolution.

Definition 10. We define $\sum\limits_{\mathrm{Tj} \in \mathrm{LPMCS(Ti)}} (\ |\ P(\mathrm{T}_i) - P(\mathrm{T}_j)\ |\ )$ as Real-time Influence Factor of T$_i$, notated by RIF(T$_i$).

RIF(T$_i$) reflects the severity degrees of violating the timing constraints caused by aborting T$_i$ in the conflict resolution.

4.2 Introduction of Similarity

Similarity is closely related to the important idea of imprecise computation in real-time systmes and to the idea of partial computation for database [10]. For many real-time applications, the value of a data object that models an entity in the real world cannot, in general, be perfectly precise reflection of the corresponding entity by reason of the delay of sampling the real world and writing the sampling value into databases. Therefore, instantaneous inconsistencies in limit range are usually permitted for real-time database systems, and the instantaneous inconsistencies can be recover by the update of the next sampling values. Obviously, The traditional conflict serializability criterion is too stringent for MDRTDBSs. Thus, we relax serializability criterion in our SHORTCC by introducing data similarity and operation similarity.

Definition 11. For data object D, suppose V$_1$(D) and V$_2$(D) are two values of D. If the distance between V$_1$(D) and V$_2$(D), notated by DIS (V$_1$(D), V$_2$(D)), meets the condition:

$$\mathrm{DIS}\ (\mathrm{V}_1(\mathrm{D}), \mathrm{V}_2(\mathrm{D})) \le \sigma$$

V$_1$ (D) and V$_2$ (D) are said to be similar, notated by V$_1$(D) $\approx$ V$_2$(D), where σ is threshold value that depends on the application semantics, and

$$\mathrm{DIS}\ (\mathrm{V}_1(\mathrm{D}),\mathrm{V}_2(\mathrm{D})) = |g(\mathrm{V}_1(\mathrm{D})) - g(\mathrm{V}_2(\mathrm{D}))|$$

where g is a mapping from the domain of D to the real number space.

Definition 12. Suppose that OP$_i$ and OP$_j$ are two operations of concurrent transactions T$_i$ and T$_j$ on the same data object D, respectively. If $\mathrm{V}_{\mathrm{OP}i}(\mathrm{D}) \approx \mathrm{V}_{\mathrm{OP}j}(\mathrm{D})$, then operations OP$_i$ and OP$_j$ are said to be similar, notated by $OP_i \approx OP_j$, where V_{OPi} (D) and V_{OPj} (D) are the values of D produced by the OP_i and OP_j, respectively.

Definition 13. Assume that the operations OP_1, OP_2,…, OP_n of concurrency transactions T_1, T_2, … , T_n act on the same data object D, and V_{OP1} (D), V_{OP2} (D), …, V_{OPn} (D) are respectively the values of D that are produced by the operations OP_1, OP_2,…, OP_n . If the following conditions is met:

$\forall V_{Opi}$ (D), V_{OPj} (D) $\in \{V_{OP1}$ (D), V_{OP2} (D), …, V_{Opn} (D) $\}$, DIS (V_{OPi} (D), V_{OPj} (D)) $< \sigma$
we define that the operation set $\{OP_1, OP_2, …, OP_n\}$ is similar.

Definition 14. Let SD*i* and SD*j* be two different states of the database DB, if the following condition is held:

$\forall$ D $\in$ DB ($\exists$ V*i*(D) $\in$ SD*i*, V*j*(D) $\in$ SD*j* (V*i*(D) $\approx$ V*j*(D)))

then SD*i* and SD*j* are referred to as similar, notated by SD*i* $\approx$ SD*j*, where V*i*(D) and V*j*(D) are the values of D in SD*i* and SD*j*, respectively.

Definition 15. Let SCH_a be any schedule for a transaction set ST = $\{T_1, T_2, …, T_n\}$ and SD_a be the states of the database produced by SCH_a. Iff

$$\exists SCH_b \ (SD_a \approx SD_b)$$

hold, SCH_a is defined as a similar serializable schedule (we also say SCH_a to meet similarity serializability), where SCH_b is any serial schedule for ST and SD_b is the database state produced by SCH_b.

4.3 Concurrency Control Protocol

Similar to the traditional real-time optimistic concurrency control protocols, the SHORTCC also classifies the executing process of an MDRTT into three phases: read phase, validation phase and write phase. The differences from the traditional real-time optimistic concurrency control protocols are the SHORTCC adopts the locking mechanism and meanwhile incorporates the security check mechanism at the validation.

During the read phase, all the subtransactions of an MDRTT called as optimistic subtransactions are distributed and optimistically executed on their participators FDSs, respectively. Once entering the validation phase, each of these FDSs triggers a base transaction for replacing the optimistic subtransaction, validating consistency, guaranteeing security constraints and committing the data locally.

An optimistic subtransaction *OPST* is a *5*- tuple: *OPST* **::=** (*PID, TID, SO, DS, TC*), where *PID* denotes the identifier of its MDRTT; *TID* represents the identifier of the subtransaction; *SO* denotes the sequence of the operations of the subtransaction; *DS* denotes the data set to be accessed by the subtransaction; *TC* denotes the timing constraints of its MDRTT.

For a write operation, an *OPST* just write a new value into its Write Set (WS), instead of update the database. For a read operation, the *OPST* reads the data required first from its WS and then from the database if the data required has not been in the WS, and records the value of data object into Read Set (RS).

During the execution of an *OPST*, the MH may be disconnected from any MSS and thus the coordinator will fall into waiting without result. Further, the other *OPSTs* of this MDRTT will also fall into waiting state. In order to avoid this situation to occur, when the coordinator is in waiting state, it sends a message of detecting network link state to the MH every other regular time interval. If the coordinator makes sure that the MH has been in disconnection or the MDRTT has expired its deadline, it broadcasts a message to abort the MDRTT. After the corresponding participators receive the abort message, they abort the corresponding *OPSTs* immediately.

Definition 16. Let $OPST_i$ be an optimistic subtransaction, the corresponding base transaction be BT_i and ROS be the Read Operation Set of $OPST_i$ and BT_i. Let RDS_o represent the data object set read by $OPST_i$, and RDS_b represent the data object set read by BT_i. Let RD_o (r) denote the data objects read by the read operation r of $OPST_i$, and RD_b(r) denote the data objects read by the read operation r of BT_i. We say that there exists a Conflict of Read Set (CRS) if one of the following conditions is true:

$$(1)\ RDS_o \neq RDS_b$$

$$(2)\ (\exists r \in ROS) \wedge \neg(V_o(RD_o(r)) \approx V_b(RD_b(r)))$$

where $V_o(RD_o$ (r) represents the values of the data objects read by the read operation r of $OPST_i$, and

$V_b(RD_b(r))$ represents the values of the data objects read by the read operation r of BT_i.

Once all of $OPST_i$ ($1 \le i \le m$) of an MDRTT have finished, the MDRTT enters validation stage and two-phase commit protocol is adopted. The coordinator firstly sends a message of PREPARE to all the participators, and after receiving the message, every participator triggers the relevant base transaction BT_i. The BT_i inherits the priority of the corresponding MDRTT. For the all BT_i on the same FDS, the two-phase locking high priority based on similarity combined with security check (2PLHP-2S for short) is adopted to control their concurrent executions, detect CRS and guarantee data security. For 2PLHP-2S, we design the following locks: R lock (read lock), X lock (write lock), and S lock (similarity lock) which includes SR (similarity read lock) and SW (similarity write lock). The lock compatibility matrix is shown in Table 1.

Table 1. Compatibility Matrix of Lack

Holding / Requesting	R	X	SR	SW
R	Y	N	Y	N
X	N	N	N	N
SR	Y	Y	Y	Y
SW	Y	Y	Y	Y

R locks, X locks and S locks are designed for base transactions at validation stage. When a base transaction executes read operation or write operation on certain data object *D*, it has to first apply for an R lock or X lock on *D*. If CCM (concurrency control manager) detects no operation conflict, it grants the base transaction the corresponding lock; or else if the conflict operations are similar, CCM grants the transaction the corresponding SR lock or SX lock.

Only when a low clearance level transaction is blocked by a high clearance level transaction, can the concurrency control covert channel be caused, so we give the following rules about how to decide for blocking or aborting the validating transaction at validation phase. In the following descriptions, T_i denotes the validating transaction; $BT(T_i)$ denotes T_i is blocked; $AT(T_i)$ denotes T_i is aborted; ω denotes the weight of satisfying security constraints and ($1-\omega$) denotes the weight of satisfying timing constraints. The value of ω can be adjusted dynamically according to the requirements of applications.

Rule 1. If the condition $((\exists T_j \in MCS(T_i)\ (P(T_i) < P(T_j)) \wedge (\exists T_j \in MCS(T_i)\ (C(T_i) < C(T_j))) \wedge (\omega \times SIF(T_i) \ge (1-\omega) \times RIF(T_i)))$ holds, $AT(T_i)$.

Rule 2. If the condition $((\exists T_j \in MCS(T_i)\ (P(T_i) < P(T_j)) \wedge (\exists T_j \in MCS(T_i)\ (C(T_i) < C(T_j))) \wedge (\omega \times SIF(T_i) < (1-\omega) \times RIF(T_i)))$ holds, $BT(T_i)$.

Let BTS denote the set of base transactions at a FDS, $T_i \in$ BTS and $MCS(T_i) = \{T_{i,1}, T_{i,2}, \ldots, T_{i,m}\}$. Suppose that T_i is requesting a lock on data object *D*, and each $T_{i,k}$ ($k=1,2,\ldots,m$) has held the lock on data object *D*. $OP(T_i)$ denotes the T_i's operation of requesting the lock on *D* and $OP(T_{i,k})$ denotes the $T_{i,k}$'s operation of holding the lock on *D*. We use $PCS(T_i)$ to denote the operation set conflicting with $OP(T_i)$, i.e. $PCS(T_i) = \{OP(T_{i,1}),\ldots, OP(T_{i,m})\}$. The 2PLHP-2S may be described as follows:

```
IF  ($P(T_i) \ge \max(P(T_{i,k}))$)  // k=1, 2 ,..., m
  {
    IF ( $\forall OP(T_{i,k}) \in PCS(T_i)$ ( $OP(T_{i,k}) \approx OP(T_i)$))
      $T_i$ obtains the S lock that $T_i$ is requesting;
    ELSE
      $T_i$ obtains the corresponding lock (R lock or X
        lock), and $T_{i,k}$ is terminated. If $T_{i,k}$ is a base
          transaction of an MDRTT, the abort
          message is sent to the coordinator of the
          MDRTT and the coordinator decides to
          terminate the MDRTT permanently or to
          restart the MDRTT according to its
          deadline;
  }
ELSE
  {
    IF ( $\forall OP(T_{i,k}) \in PCS(T_i)$ ( $OP(T_{i,k}) \approx OP(T_i)$))
      $T_i$ obtains the S lock that $T_i$ is requesting;
    ELSE
    {
        IF ($\omega \times SIF(T_i) \ge (1-\omega) \times RIF(T_i)$)
        Abort $T_i$ and send the abort message to the
              coordinator;
      ELSE
        Block $T_i$;
    }
  }
```

During a base transaction execution, for every read operation, the MRTM (mobile real-time transaction manager) validates if any CRS has been happened. If a CRS is detected, the MRTM aborts the base transaction and sends a message "Non-OK" to the coordinator. After receiving the message, the coordinator decides to abort the corresponding MDRTT and broadcasts the decision to all the related participators of the MDRTT, and then releases the corresponding system resources. After aborting an MDRTT, the MRTM may decide to restart it if possible, instead of permanently terminating it. If a base transaction passes the validation, the corresponding participator sends a message "OK" to the coordinator. After receiving an "OK" from all the participators, the coordinator sends a message of Global Commit to all the participators. When a participator receives the message of Global Commit, the corresponding base transaction finishes the write phase and sends a message of ACK to the coordinator.

In the below, we prove that the SHORTCC can guarantee a schedule to meet similarity serializability. Let SCH be any schedule of a transaction set in the system. There exists the following theorem:

Theorem 1. If SCH obeys SHORTCC, SCH is a similar serializable schedule.

Proof: (1) Suppose G = (V, E) is the precedence graph of SCH where V = {T_1 , T_2 ,…, T_n }, E = {($T_i \rightarrow T_j$) | T_i, $T_j \in$ V, OP_i and OP_j which belong to T_i and T_j respectively are a pair of conflict operations and OP_i is executed before OP_j}. Let G^1 = (V, E^1) is the directed graph that is gotten by throwing away the directed edges of G that are caused by conflict operations of similarity. Obviously, G^1 is acyclic according to 2PL (two-phase locking).

(2) Take out a directed edge e from (E - E^1) and join e into G^1. Obviously, e is caused by a pair of conflict operations of similarity. Suppose this pair of conflict operations are OP_k from T_k and OP_m from T_m, namely e = ($T_k \rightarrow T_m$). If G^1 becomes cyclic, owing to $OP_k \approx OP_m$, the database state caused by exchanging the executing order of OP_k and OP_m is similar with the original database state. So the ring in G^1 can be eliminated and thus the similarity of the database state is assured.

(3) Repeat step (2), until (E - E^1) becomes empty.

(4) Through the above steps, we can assure that the final G^1 is acyclic. Suppose the certain schedule corresponding to the final G^1 is SCH^1. Obviously, SCH^1 is conflict serializablity.

(5) Because the database state produced by SCH is similar to the database state produced by SCH^1, SCH is a similar serializable schedule.

5. Experiments and Results

The simulation experiments are based on ARTs-II, which is a distributed real-time database system test bed developed by our lab. We improve ARTs-II in order to support mobile distributed real-time transactions. In our simulation system, each MH has a transaction generator, a transaction manager, a message server, a handoff handler and a disconnection handler. Each FDS is a local database system consisting of a scheduler, a concurrency control manager, a message server, a ready queue, a block queue and a local database (LDB). The global database (GDB) is modeled as a collection of data pages that are uniformly distributed across all the sites. Transactions make requests for data pages and concurrency control is implemented at page level. At each node, transactions arrive in an independent Poisson rate. Each transaction is modeled as a sequence of operations. The processing of an operation involves use of the CPU and access to data items. Transactions are queued in the ready queue for CPU. The queuing discipline is earliest deadline first (EDF). The deadline of a transaction T is set using the following formula: *Deadline* (T) = *AT*(T) + Slack × *ET*(T), where *AT*(T) is the arrival time of T; Slack is the slack time assigned to T, which is a uniformly distributed random variable within a specified range; *ET*(T) denotes an estimated execution time of T. The baseline setting of the values for parameters is shown in Table 2.

Table 2. Model parameters and their baseline values

Parameters	Baseline Values	Descriptions
NMH	10	Number of mobile hosts
NFH	5	Number of fixed hosts
NMSS	5	Number of mobile support bases
LDB	200 pages	Size of LDB at each FDS
PDis	0.05	Probability of disconnect
PHoff	0.02	Probability of handoff
TS	4 to 6 operations	Transaction size
Rate	[5, 40]	Arrival rate of transactions
P_U	0.4	Probability of update operations
ThinkTime	0	Time interval which MH waits for transmitting the next transaction after the former has committed
Slack	U [2.0, 6.0]	Slack factor which is a uniformly distributed random variable in the range [2.0, 6.0]
NCL	6	Number of different clearance levels (level 1~level 6)
RDCL	1/6	Ratio of different clearance level transactions
P_S	0.4	Similarity probability between operations of different transactions

In our experiments, we firstly compare the real-time performance of our SHORTCC with that of the real-time concurrency control protocol DHP2PL (the distributed version of HP2PL). Then, we study how the different values of ω (the weight of satisfying security constraints) influence data security for SHORTCC at fixed arrival rate of transactions. The main performance metrics used for the evaluations are the ratio of transactions missing their deadlines denoted as *TMDR*, which is defined as the number of deadline-missing transactions over the total number of transactions generated in the system, and total average number of low clearance level transactions blocked by high clearance level transaction per 5 seconds, notated as *NLCB*. Here, *TMDR* reflects the real-time performance, and *NLCB* reflects the severity degree of violating the security constraints. Further, we use *HCTMDR* to denote the *TMDR* of high clearance level transactions, which is defined as the number of deadline-missing low clearance level transactions over the total number of transactions generated in the system, and *LCTMDR* to denote the *TMDR* of low clearance level transactions, which is defined as the number of deadline-missing high clearance level transactions over the total number of transactions generated in the system. Here, high clearance level includes level 4, level 5 and level 6, and low clearance level includes level 1, level 2 and level 3. Obviously, there exists the following equation: *TMDR* = *HCTMDR* + *LCTMDR*.

The performance results are shown in Fig.1–Fig.4. Fig.1 illustrates the ratio of low clearance level transactions missing their deadlines (*LCTMDR*) for DHP2PL and SHORTCC at ω=0.5. When arrival rate of transactions is below 25, The DHP2PL slightly gets an advantage over the SHORTCC. The reason is that for SHORTCC, when a low clearance level transaction conflicts with a high clearance level transaction, the low clearance level transaction is usually aborted and restarted in order to be free from the concurrency control covert channel. However, after an arrival rate of 25, The SHORTCC has an advantage over DHP2PL slightly. This can be illustrated by that when arrival rate of transactions increases, many conflict operations between low clearance level transactions and high clearance level transactions may be similar, thereby avoiding abort and restarting. As shown in Fig.2 (ω=0.5), when arrival rate of transactions enhances, *HCTMDR* of both DHP2PL and SHORTCC increase, but the performance of SHORTCC obviously gets an advantage over DHP2PL. The main reasons are the introduction of similarity serializability and high clearance level transactions will not be aborted due to the concurrency control covert channel. Fig.3 illustrates the total average number of low clearance level transactions blocked by high clearance level transaction per 5 seconds (*NLCB*) for DHP2PL and SHORTCC at ω=0.5. Obviously, SHORTCC excels DHP2PL prominently in *NLCB*. From Fig.1- Fig.3, we can know that the SHORTCC achieves data security without sacrificing real-time performance significantly.

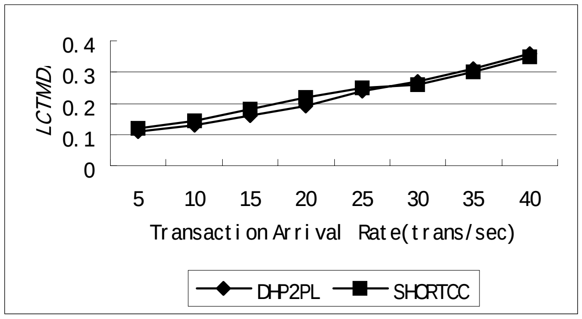

Fig.1. *LCTMDR* comparison for DHP2PL and SHORTCC

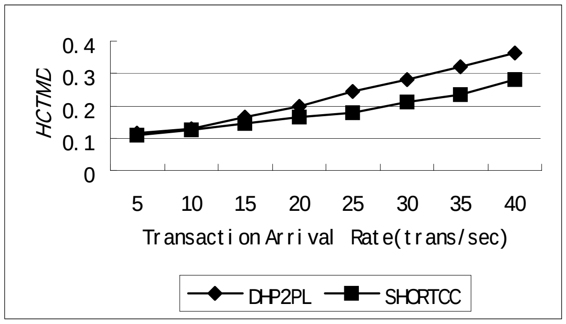

Fig.2. *HCTMDR* comparison for DHP2PL and SHORTCC

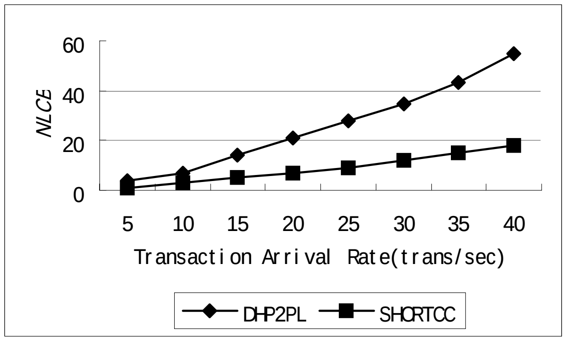

Fig.3. *NCLB* comparison for DHP2PL and SHORTCC

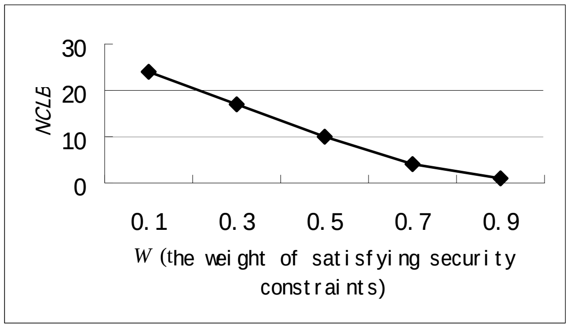

Fig.4. Influence on *NCLB* of different values of ω

Fig.4 illustrates how ω influences the *NLCB* of SHORTCC at the fixed arrival rate of transactions (25trans/sec). As shown in Fig.4, the *NLCB* increases with the enlargement of ω.

6. Conclusion

A secure mobile distributed real-time database system has to simultaneously satisfy the two goals of guaranteeing data security and minimizing the deadline miss ratio for admitted transactions. However, these two goals can conflict with each other and to achieve one goal is to sacrifice the other. This paper presents a secure hybrid optimistic real-time concurrency protocol (SHORTCC). Unlike the traditional optimistic concurrency control protocols, the SHORTCC adopts locking mechanism to guarantee data consistency and introduces security check to achieve data security at the validation phase. In order to improve the concurrency, the SHORTCC relaxes serializability criterion by introducing data similarity and operation similarity. During the security check, if exists the possibility of violating the security constraints, the decision of blocking or aborting the validating transaction is made by comparing the total severity degrees of violating the security constraints, which are caused by blocking the validating transaction, with the total severity degrees of violating the timing constraints, which are caused by aborting the validating transaction. According to the application requirement, the SHORTCC can make a suitable tradeoff between security and real-time performance by adjusting the value of ω (weight of satisfying security constraints). Simulation experiments show that the SHORTCC not only achieves data security, but also guarantees high real-time performance and does a favor to meet the deadline requirement of transaction in mobile environment.

References

[1] C.L. Liu, J.L. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment [J]. *Journal of ACM*, 1973, 20(1): 46-61.

[2] R. Abbott, H. Carcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation [J]. *ACM Transactions on Database Systems*, 1992: 513-560.

[3] L. Sha, R. Rajkumar, S.H. Son, et al. A real-time locking protocol [J]. *IEEE Transactions on Computer*, 1991, 40(7): 793-800.

[4] J. Haritsa, M. Carey, M. Livny. Data access scheduling in firm real-time database systems [J]. *Real-Time Systems Journal*, 1992, 4(3): 203- 241.

[5] R. Abbott, H. Garcia-Molina. Scheduling real-time transactions: a performance evaluation [J]. *ACM Transactions on Database Systems*, 1992, 17(3): 513-560.

[6] J.R. Haritsa, M. Livny, M.J. Carey. On being optimistic about real-time constraints. *In Proceedings of the 9th ACM Symposium on Principles on Database Systems*, pp. 313-343 (1990).

[7] T.W. Kuo, A.K. Mok. SSP: a semantics-based protocol for real-time data access. *In Proceedings of IEEE 14th real-time systems symposium*, pp. 76-86 (1993).

[8] 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).

[9] O. Ulusoy, G.G. Belford. Real-time transaction scheduling in database systems [J]. *Information Systems*, 1993, 18(8): 559-580.

[10] K.Y Lam, T.W Kuo, W.H. Tsang, et al. Concurrency control in mobile distributed real-time database systems. *Information Systems*, 2002, 25(4): 261-286.

[11] K.Y. Lam, W. Yau. On using similarity for concurrency control in real-time database systems. *Journal of Systems and Software*, 1998, 43(3): 223-232.

[12] S.H Son, C. Chaney, N. Thomlinson. Partial security policies to support timeliness in secure real-time databases. *In Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 3-6 May 1998.

[13] S.H. Son, R. David, C. Chaney. Design and analysis of an adaptive policy for secure real-time locking protocol [J]. *Journal of information Science*, 1999, 99(1): 101-135.

[14] Q. Ahmed, S. Vrbsky. Maintaining security in firm real-time database systems. *In Proceedings of Computer Security Applications*. Phoenix, AZ, USA, 7-11 Dec. 1998.

[15] B. George, J. Haritsa. Secure transaction processing in firm real-time database systems [J]. *SIGMOD Record*, 1997, 26(2): 462-473.

[16] R. David, S.H. Son, R. Mukkamala. Supporting security requirements in multilevel real-time databases. *In Proceedings of IEEE Symposium on Security and Privacy*. Oakland, CA, USA, 8-10 May 1995.

[17] S.H. Son, R. David, T. Bhavani. An adaptive policy for improved timeliness in secure database systems. *In Proceedings of Annual IFIP WG 11.3 Conference of Database Security*, Aug. 1995.

[18] S.H. Son, R. Mukkamala, R. David. Integrating security and real-time requirements using covert channel capacity[J]. *IEEE Transactions on Knowledge and data Engineering*, 2000, 12(6): 865-879.

[19] N.A. Quazi, V.V. Susan. Maintaining security and timeliness in real-time database system [J]. *Journal of Systems and Software*, 2002, 61(1): 15-29.

[20] Bell D, LaPadula L. Secure computer systems: Unified Exposition and Multics Interpretation. *Technical Report MTIS AD-A023588*, *The Mitre Corp*, July 1975.