Concurrency Control in Mobile Real-Time Database System (MRTDBS)

Nyo Nyo Yee, Hninn Aye Thant
University of Technology (Yatanarpon Cyber City)
nny1ster@gmail.com, hninayethant@gmail.com

Abstract

In mobile environment, transactions are initiated from mobile host and that may be executed at mobile host or fixed host. This system proposes Two-Shadow Speculative Concurrency Control (SCC-2S) with Priority mechanism for Mobile Real Time Database System (MRTDBS). Proposed system is designed for strict deadline nested transactions. Concurrency Control will perform at the Fixed Host and the results are returned back to the mobile hosts. SCC-2S is based on Speculative Concurrency Control (SCC). SCC allows a large number of shadows for each uncommitted transactions in the system to co-exist. SCC-2S uses at most two shadows for each transaction, primary shadow and standby shadow. In order to become new contribution, we add priority control mechanism and solve write/write conflict in existing SCC-2S strategy.

Keywords - concurrency control; mobile real-time database system; nested transaction; fixed host

Related work

In mobile database environment, providing consistency of the data items is a challenging problem in case of concurrent access. Conventional two phase locking protocol is not suitable, as it requires clients to communicate continuously with the server to obtain locks and detect the conflicts[6]. M. K. Nizamuddin1 [5] et. al. proposed new priority based scheme in which priority is given to the older transaction whenever conflict arises, so that this method decrease transaction re-executions and current load of the database server. But Invalidation Report (IR), Absolute Validity Interval (AVI) are required. This method applies for only flat transactions and mobile host require frequent connection with fixed host. A. Karami [1] et. al. proposed the OPCOT concurrency control algorithm based on optimistic concurrency control method. This method reduces communications between mobile client and server, decrease blocking rate and deadlock of transactions. To reduce abortion rate of transactions, mobile client is assigned timestamp for each transaction operators. Database server used this timestamp to determine the time of commitment. The disadvantages of this method are overhead of timestamps and their calculation (relative time, absolute time for each operator).

1. Introduction

Mobile Real-time Database System (MRTDBS) provide information to a mobile host (mobile user). Primary objective for MRTDBS is to minimize the number of missed deadlines. Mobile host can initiate transactions from anywhere and at anytime. When shared data item is updated by multiple transactions at the same time, Concurrency Control(CC) techniques become critical to guarantee timely access and correct results (Consistency) for concurrent mobile transactions. General characteristics of mobile environments like mobility, low bandwidth, limited battery power, limited storage, frequent disconnections etc. makes concurrency control more difficult [9].

Various concurrency control algorithms differ from the time when conflicts are detected, and the way they are resolved. Pessimistic Concurrency Control (PCC) and Optimistic Concurrency Control (OCC) alternatives differ from data conflict detection and conflict resolution. PCC locking protocols detect conflicts as soon as they occur and resolve them using blocking, while OCC protocols detect conflicts at transaction commit time and resolve them using restarts.

Speculative Concurrency Control (SCC) algorithms combine the advantages of both PCC and OCC algorithms, and avoid their disadvantages. SCC resembles PCC in that potentially harmful conflicts are detected as early as possible, and thus it increases the chances of meeting timing constraints. Moreover, SCC resembles OCC in that it allows conflicting transactions to proceed concurrently, thus it avoids unnecessary delays to meet timely commitment. SCC allows many shadows for uncommitted transactions. But, SCC-2S allows a maximum of two shadows per uncommitted transaction to exist in the system at any point in time: a primary shadow and a standby shadow [2]. The primary shadow means the original nested transaction query to access shared data. The standby shadow means the copy of the original query that does not contain the portion of the query that the primary shadow is already performed.

The rest of this paper is organized as follows: Section 2 briefly introduce mobile database environment. In section 3, we present our proposed method and proposed system architecture. In section 4, we present mathematical expression of our proposed method. Section 5 draws the conclusion.

2. Mobile Database Environment

Mobile database environment consists of Mobile Host (MH), Fixed Host (FH) and Base station (BS). The communication of the MH and FH is supported by Base station. FH and BS are connected with a wired network and FH has a Database server. Some MH have Database Management System (DBMS) module to perform database operations. In mobile environment, MH can process its workload in continuously connected mode or in disconnected mode or in intermittent connected mode.

In connectivity mode, MH is continuously connected to the database server. MH has cache to store the required data. Moreover, MH can request data from the server any time during transaction processing.

In disconnected mode, MH voluntarily disconnects from the server after refreshing the cache and continues to process workload locally. At a fixed time, it connects and sends its entire cache to the server. The server installs the contents of the cache to maintain global consistency.

Intermittent connected mode is similar to disconnected mode, but the MH can be disconnected any time by the system or voluntarily by the user [3].

There are three type of data dissemination mode in mobile environments. In Broadcast Mode (push process), broadcast server periodically broadcast most popular data on some wireless channel form which users can listen and, download if the data is required. There is no uplink channel involved in this mode. In On-Demand Mode (pull process), client request specific data which is not available in the current broadcast or may never appear in the broadcast. Client send the query for require data through an uplink channel. In Hybrid Mode, broadcast and on-demand modes are combined [8].

2.1. Classification of Transactions

There are three types of transactions used in database system. They are flat transactions, nested transactions and distributed transactions. All of these transactions have four properties. These properties are Atomicity, Consistency, Isolation (Independence) and Durability (or Permanency).

Flat transactions access a single database and adequate for simple applications. During of their execution, they do not trigger other dependent transaction [8]. Nested Transactions are constructed from a set of sub-transactions. Each sub-transaction may also have sub-transactions, and nesting can occur to arbitrary depth. "Figure. 1" define nested transaction model.

There are two types of Nested transaction model, open nested transaction model and closed nested transaction model. In open nested transaction model, the parent transaction does not enforce restriction of the execution, rollback, and commitment of its sub transactions. The parent transaction only invokes sub-transaction, and the sub-transactions executed independently to each other and to the parent. Locks are released before parent transaction. But, in closed nested transaction model, locks are released after parent transaction [11].

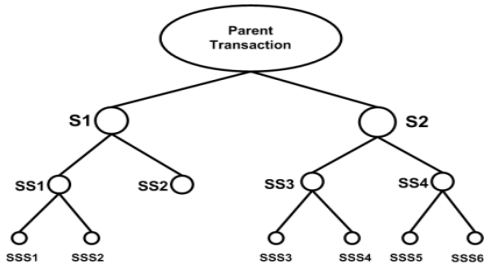

Figure 1. Nested transaction Model

Distributed transactions concern with a physical division of transactions due to the need of accessing distributed resources. Special distributed algorithms are needed to handle locking of data and committing of transactions [10].

2.2. Execution Modes in Mobile Environment

Transactions are initiated at mobile host but may be executed on mobile host or fixed host or the execution may be distributed between mobile host and fixed host respectively. There are five execution modes in Mobile Environment. They are

• Complete Execution on Fixed Network Transaction is initiated at mobile host but is completed executed at fixed network. In this approach, mobile host acts as a thin client.

• Complete Execution on MH

Transactions are initiated at mobile host and are executed on mobile host. This approach requires mobile hosts to have processing and storage capabilities as well as managing data. But, reconciliation is needed with fixed host at some point in time.

• Distributed Execution on MH and Wired Network

Transaction is initiated at mobile hosts and the execution is distributed among mobile host and fixed host. A sub-transaction is executed at fixed host and another one at mobile host. This approach helps in minimizing the communication between the fixed host and mobile hosts.

• Distributed Execution among several MHs

Transaction is distributed among several mobile hosts for execution. It provides a peer-peer strategy. A mobile host acts as a server for other mobile hosts so that the execution is distributed between them. The selection of a mobile host for execution of a transaction is location based.

• Distributed Execution among MHs and FHs

Transaction execution is distributed among several mobile hosts and fixed hosts respectively. In this approach, multiple parties may be involved [4].

3. Proposed System

In our system, mobile user sends query using an uplink channel (pull process). To process the request, database server in Fixed Host use proposed method

Two-Shadow Speculative Concurrency Control (SCC-2S) with Priority that avoids conflict (access the same data). To increase the degree of parallelism in the execution of long running transaction is primary objective of nested transaction. This system aim for strict deadline nested transactions to improve inter-transaction concurrency. Mobile Host (MH) can live intermittent connected mode (not need to connect the database server) when Fixed Host (FH) perform database operation. After the database operation had performed, FH returns the result back to corresponding MH. MH does not require having Database System (DBMS) module to perform database operations. So, MH acts as a thin client. We define our proposed system for air defence system.

MH sends air craft data together with deadline to the database server located in Fixed Host (FH). Database server determine which type of weapon can be used if the aircraft is enemy's aircraft. And then FH send the decision to the dedicate MH. In FH, the database server uses our proposed method Two-Shadow Speculative Concurrency Control (SCC-2S) with Priority to control concurrent access. When two or more nested transactions from two or more MHs enter our system simultaneously, if their deadlines are not equal, the transactions with earliest dead line enter the system first. If the two transactions have the same deadline, enter our system concurrently. During the transactions run, if conflicts occur between the transactions, SCC-2S with priority algorithm is used to solve the conflict. When the two transactions encounter (read/write) conflict, the read transaction create the standby shadow (copy the transaction) where the conflict is detected. This standby shadow excludes the part of the transaction that the primary shadow (original transaction) already performed. When the two transactions encounter (write/write) conflict, the transaction with late time creates the standby shadow. But the two transactions encounter (write/write) conflict at the same time, our proposed system uses priority control mechanism to determine which transaction should create the standby shadow.

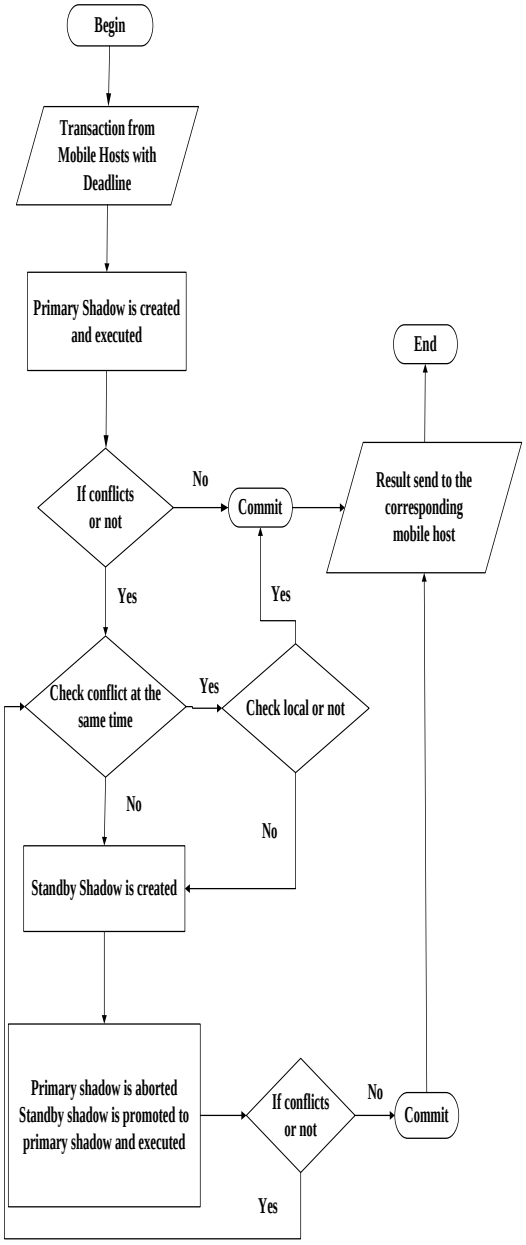

Figure 2. Flow diagram for proposed system

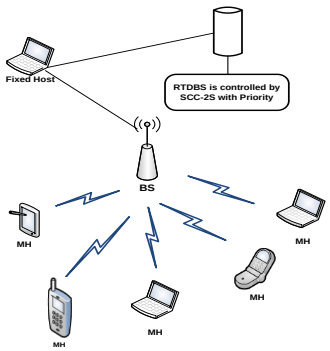

Figure 3. Proposed system architecture

3.1. Priority Control Mechanism

In this theory, we define T_{rt} as lock-requesting transaction and T_{ht} as lock-holding transaction. Transaction is local if it access data items available at only one Database Module. Otherwise it is a global transaction. If transaction access only one database it is assigned high priority otherwise it is assigned low priority.

```
Lock Conflcit (Trt, Tht)
Begin
   If  Priority(Trt) > Priority(Tht)
       If  Tht is not committing
           If  Tht is a local transaction
              Create standby shadow Tht locally
           Else
             Create standby shadow Tht
globally
           Endif
       Else
            Block Trt until Tht releases the lock
            Priority(Tht):=Priority(Trt)+fixed priority
            level
       Endif
   Else
         Block Trt until Tht releases the lock
   Endif
End[7].
```

3.2. Two Shadow Speculative Concurrency Control (SCC-2S) Algorithm with Priority

Speculative Concurrency Control (SCC) is especially designed for real-time database applications. It relies on the use of redundancy to ensure that serializable schedules are discovered and adopted as early as possible, that increase in timing commitment of transactions with strict timing constraints. Two-Shadow SCC algorithm (SCC-2S), a member of the SCC-nS family, and minimal use of redundancy. SCC-2S uses at most two shadows for each transaction, primary shadow and standby shadow. Original SCC-2S algorithm solves read/write conflict for concurrent transactions. For our contribution, we want to solve write/write conflict for nested transaction and also add priority control mechanism for nested transaction. The detailed explanation for SCC-2S is:

Let T_j be any uncommitted transaction in the system. The primary shadow for T_j runs under the optimistic assumption that it will be the first (among all the other transactions with which T_j conflicts) to commit. Therefore, it executes without incurring any blocking delays. At that time, standby shadow for T_j, is subjected to blocking and restart. It is kept ready to replace the primary shadow, if replacement is needed.

The SCC-2S algorithm resembles Optimistic Concurrency Control with Broadcast Commit (OCC-BC) algorithm in that primary shadows of transactions continue to execute either until they validate or commit or until they are aborted. The difference is that SCC-2S keeps a standby shadow for each executing transaction to be used if that transaction must abort. The standby shadow is basically a replica of the primary shadow, except that it is blocked at the earliest point where a Read-Write conflict is detected between the transaction it represents and any other uncommitted transaction in the system. If required, the standby shadow is promoted to become the primary shadow, and execution is resumed [4].

Illustration of SCC-2S works is, shown in "Figure. 4", The two mobile units MU1 and MU2 access the same data item x. MU1 execute Transaction T_1 to write data item x. MU2 execute Transaction T_2 to read data item x. Both transactions T_1 and T_2 start with one primary shadow, namely T_1^0 and T_2^0 respectively. When T_2^0 attempts to read object x, a potential conflict is detected. At this point, a standby shadow, T_2^1, is created. The primary shadows T_1^0 and T_2^0 execute without interruption, whereas T_2^1 blocks. Later, if T_1^0 successfully validates and commits on behalf of transaction T_1, the primary shadow T_2^0 is aborted and replaced by T_2^1, which resumes its execution, we hope to commit before its deadline [3] .

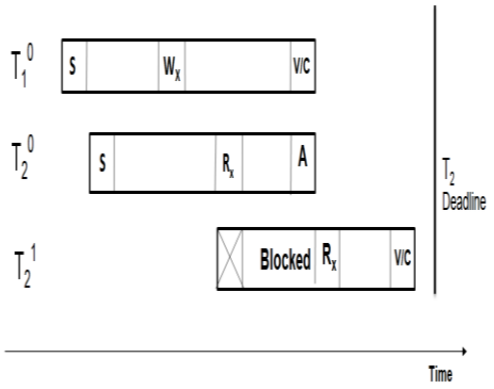

Figure 4. Schedule with a standby shadow promotion

It is possible that multiple conflicts develop between executing transactions. The three mobile units MU1, MU2 and MU3 access the same data. MU1 execute Transaction T_1 to write data item y. MU2 execute Transaction T_2 to read data item x and data item y. MU3 execute Transaction T_3 to write data item x. "Figure. 5" illustrates the behavior of SCC-2S when a second conflict develops between T_2 and another transaction T_3. In particular, the primary shadow T_1^0 of T_1 attempts to write an object y that both shadows T_2^0 and T_2^1 had previously read. In this case, the primary shadow T_2^1 create the standby shadow T_2^2 to solve conflict with transaction T_1.

The SCC-2S algorithm allows at most two shadows for the same transaction to co-exist at any given time. In particular, after T_2^1 is promoted to become the primary shadow for T_2, a standby shadow T_2^2 is forked off to account for the read-write conflict between T_2^1 and T_1 [3].

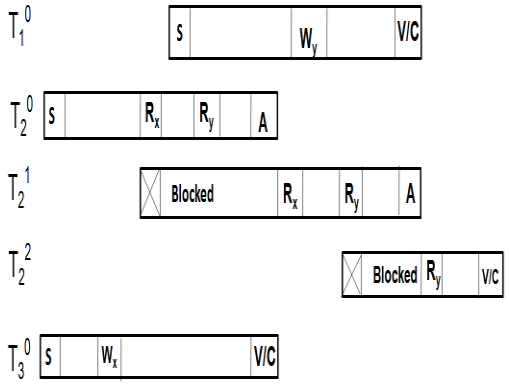

Figure 5. Schedule with two standby shadows

For our contribution, we add write/write conflict for concurrency control in "Figure. 6" The two mobile units MU1 and MU2 access the same data item x. MU1 execute transaction T_1 to write data item x. MU2 execute Transaction T_2 to write data item x. Write conflict time for transaction T_2 late. So transaction T_2 create standby shadow. But, in "Figure.7" write conflicts occur at the same time. At that time, our system use priority control mechanism. Assume transaction T_1 is local transaction and transaction T_2 is global transaction. So, the standby shadow T_2^1 is created for transaction T_2.

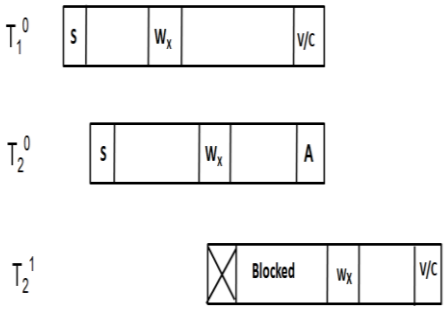

Figure 6. Schedule with a standby shadow promotion for write/write conflict

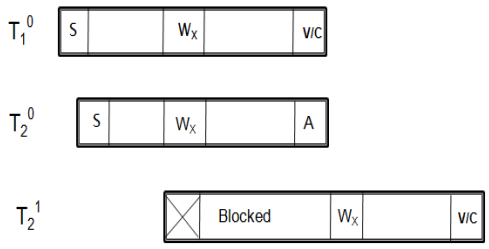

Figure 7. Schedule with a standby shadow promotion for write/write conflict using priority control mechanism

4. Mathematical Expression For Proposed Method

Let $T = T_1, T_2, T_3, \ldots, T_m$ be the set of uncommitted transactions in the system. Let $T^{primary}$ and $T^{standby}$ be primary and standby shadows executing on behalf of the transaction set T, respectively. For each standby shadow T_r^j in the system , a set WaitFor(T_r^j) is maintained , which contains a list of tuples of the form (T_s , Y), where $T_s \in T$ and Y is an object of the shared database. (T_s , Y) $\in$ WaitFor(T_r^j) implies that T_r^j must wait for T_s before being allowed to read or write object Y. The notation (T_s, -) $\in$ WaitFor(T_r^j) is used where there exists at least one tuple (T_s , Y) $\in$ WaitFor(T_r^j) , for some object Y. Details of the SCC-2S algorithm are defined as follows:

A. When a new transaction T_s is requested for execution, a primary shadow $T_s^0 \in T^{primary}$ is created and executed.

B. Whenever a primary shadow T_s^i wishes to read an object Y that has been written by another shadow T_r^j, then:

1. If there is no standby shadow for T_s , a new shadow T_s^{i+1} for T_s is created, such that WaitFor (T_s^{i+1}) = {(T_r , Y)}, otherwise
2. Let T_s^k be the standby shadow executing on behalf of T_s. If (T_r , Y) $\notin$ WaitFor(T_s^k), then WaitFor(T_s^k) = WaitFor (T_s^k) U {(T_r , Y)}.

C. Whenever a primary shadow T_s^i wishes to write an object X that has been read by another shadow T_r^j , then:

1. If there is no standby shadow for T_r , then a new shadow T_r^{j+1} for T_r is created and executed, such that WaitFor(T_r^{j+1}) = {(T_s ,X)}, otherwise
2. Let T_r^k be the standby shadow executing on behalf of T_r. If (T_s ,X) $\notin$ WaitFor(T_r^k), then T_r^k is aborted and a new standby shadow T_r^{k+1} is started with WaitFor(T_r^{k+1}) = WaitFor(T_r^k) U {(T_s , X)}.

D. A standby shadow T_s^i is blocked whenever it wishes to read any object that has been written on behalf of any of the transactions in WaitFor(T_s^i).

E. Whenever it is decided to commit a primary shadow T_s^i on behalf of transaction T_s , then

1. If (T_s , -) $\in$ WaitFor(T_r^j) then the primary shadow of T_r is aborted, T_r^j is promoted to become a primary shadow of T_r, and a new backup shadow T_r^{j+1} is forked off T_r^j, such that WaitFor (T_r^{j+1}) = WaitFor(T_r^j) - {(T_s , -)}.
2. Any standby shadow of T_s is aborted[2].

For contribution, we add mathematical expression for write/write conflict.

F. Whenever a primary shadow T_r^i wishes to write an object X that has been written by another shadow T_s^j, then:

1. If there is no standby shadow for T_r^i , then a new shadow T_r^{i+1} for T_r is forked off, such that WaitFor(T_r^{i+1}) = {(T_s ;X)}, otherwise
2. Let T_s^k be the standby shadow executing on behalf of T_r . If (T_s;X) $\notin$ WaitFor(T_r^k), then T_s^k is aborted and a new standby shadow T_r^{k+1} is started with WaitFor(T_r^{k+1})= WaitFor(T_r^k) U {(T_s;X)}.

5. Conclusion

Two Shadow Speculative Concurrency Control (SCC-2S) with Priority is a powerful mechanism for concurrency control in Mobile Real-time Database System (MRTDBS). SCC-2S with Priority provides high respond time and throughput. SCC-2S with priority relies on redundancy to ensure that serializable schedules are discovered and adopted as early as possible, thus increasing the likelihood of the timely commitment of transaction with strict timing constraints. SCC-2S with Priority decreases the number of missed deadlines in the system. In SCC-2S with Priority, shadow transactions execute on behalf of a given uncommitted transaction so as to protect against the hazards of blockages and restarts.

References

[1] Ali Karami , Ahmad Baraani-Dastjerdi, "A Concurrency Control Method Based on Commitment Ordering in Mobile Databases", International Journal of Database Management Systems (IJDMS) *Vol.3, No.4*, November 2011.

[2] Azer Bestavros, "Speculative Concurrency Control" ,Computer Science Department, Boston University, Boston, 02215, January 27, 1993.

[3] Azer Bestavros , Spyridon Braoudakis , Euthimios Panagos , "Performance Evaluation of Two-Shadow Speculative Concurrency Control", Computer Science Department, Boston University, Boston, 02215, February 5, 1993.

[4] Jun Chen, Yu Fen Wang, Jian Ping Wang," Concurrency Control Protocol for Real-Time Database and the Analysis Base on Petri Net", Jun Chen et al., 2010, Advanced Materials Research, 143-144, 12, October, 2010.

[5] Mohammed Khaja Nizamuddin1, Dr. Syed Abdul Sattar, "An Improved, Prioritized Concurrency Control Scheme with Performance Gain in Mobile Environments" , ARPN Journal of Systems and Software, 2010-11 AJSS Journal, *Volume 1 No. 1*, APRIL 2011.

[6] Salman Abdul Moiz, Dr. Lakshmi Rajamani, "An Algorithmic approach for achieving Concurrency in Mobile Environment", INDIACom, 209-211, 2007.

[7] Salman Abdul Moiz, Supriya N.Pal, Jitendra Kumar3, Lavanya P, Deepak Chandra Joshi, Venkataswamy G , " Concurrency Control In Mobile Environments: Issues & Challenges", International Journal of Database Management Systems (IJDMS) *Vol.3, No.4*, November 2011.

[8] Syed Abbas Bukhari , Samuel Rivera Aparicio , "A Survey of Current Priority Assignment Policies (PAP) and Concurrency Control Protocols (CCP) in Real-Time Database Systems RTBDS". MS Bioinformatics, Sustainable and Resilient Infrastructures Systems (CEE), 2012.

[9] Vishnu Swaroop, Gyanendra Kumar Gupta, Udai Shanker, "Issues In Mobile Distributed Real Time Databases: Performance And Review", India, 2011.

[10] "Transaction and Concurrency Control",CS 141b-Distributed Computation Laboratory, http://www.cs.caltech.edu/~cs141/, February 19, 2004.

[11] Hamzeh Khazaei, "Mobile Database System", Math & Computer Science Department, Amirkabir University of Technology, (Tehran Polytechnic), Hamzeh.khazaei@aut.ac.ir