Concurrency control in real-time broadcast environments

Dang Depeng *, Liu Yunsheng

College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China

Received 5 October 2001; received in revised form 15 February 2002; accepted 23 March 2002

Abstract

Owing to the unique characteristics of real-time broadcast environments, serializability is too strong as a correctness criterion and not suitable for mobile real-time transactions. Considering that relaxing serializability such as epsilon and similarity serializability may sacrifice database consistency to some extent, we propose using a correctness criterion called weak serializability. In this paper, we formally define weak serializability at first. After the necessary and sufficient conditions for weak serializability are shown, corresponding concurrency control protocol based on this criterion is outlined for real-time broadcast environments. Finally, in a series of simulation studies, experimental results show that the proposed protocol helps more mobile real-time transactions to meet their deadlines and improves response time while database consistency is maintained.

Keywords: Data broadcast; Mobile real-time transactions; Real-time concurrency control

1. Introduction

Recent advances in electronic technologies such as portable computers and wireless communication networks have led to the reality of distributed mobile computing systems (Acharya et al., 1995; Daniel, 1999). Examples of these applications are mobile auctions, telemedicine, real-time traffic information and navigation, stock trading and automated industry plants (Lam et al., 1999). Many of these applications are inherently of real-time nature (Daniel, 1999; Lee et al., 1999; Stankovic et al., 1999). For instance, it may be a financial or opportunity loss if a stock trading transaction cannot be completed within a certain deadline (Lee et al., 1999). In addition, the temporal validity of some data objects such as stock price or sensor data poses another type of timing constrains to the database system. Transaction correctness is then defined as meeting its timing constraints and using data that is absolutely or relatively timing consistent (Lee et al., 1999; Stankovic et al., 1999).

Broadcast based data dissemination becomes a popular method of data communication in mobile computing systems. In a broadcast system, the server broadcasts data periodically on the broadcast channel, and clients tune in to receive the broadcast data and filter the required data (Acharya et al., 1995; Acharya et al., 1996). The distinguished feature of the broadcast mode is the limitation and asymmetry in communication capacity. The “upstream” (client-to-server) communication capacity is relatively much smaller than the “downstream” (server-to-client) communication capacity, while even the bandwidth in the down stream direction is limited (Daniel, 1999; Imielinski and Badrinath, 1994). The limited capacity of bandwidth available in such environments places a new challenge to implementing transaction processing efficiently (Daniel, 1999; Lee et al., 1999).

Data management in broadcast environments receives a lot of attention in these few years. However, there are only a few studies on transaction processing in the area (Acharya et al., 1996; Daniel, 1999; Lee et al., 1999). Owing to the unique characteristics of real-time broadcast environments, serializability is too strong as a correctness criterion and not suitable for mobile...
real-time transactions (Daniel, 1999; Lam et al., 1999; Lee et al., 1999; Ulusoy, 1998). Considering that relaxing serializability such as epsilon and similarity serializability may sacrifice database consistency to some extent (Lam et al., 1999; Ramamritham and Calton, 1995), we propose using a correctness criterion called weak serializability. In this paper, we formally define weak serializability at first. After the necessary and sufficient conditions for weak serializability are shown, corresponding concurrency control protocol based on this criterion is outlined for real-time broadcast environments. Finally, in a series of simulation studies, experimental results show that the proposed protocol helps more mobile real-time transactions to meet their deadlines and improves response time while database consistency is maintained.

2. Related work

Broadcast-based data dissemination has been studied extensively over a few years (Acharya et al., 1995; Acharya et al., 1996). In a broadcast disk model, the server continuously and repeatedly broadcasts all objects in the database. The mobile clients view this broadcast as a disk and mobile transactions can read the values of data objects being broadcast. A periodic broadcast program is constructed to schedule the broadcast of data objects cyclically (broadcast cycle) according to certain popularity criteria (Acharya et al., 1995; Lee et al., 1999).

Typical applications in broadcast environments have clients that execute read-only transactions (queries), and a server that executes update transactions (Pitoura and Chrysanthis, 1999). The large population of read-only transactions makes the processing of read-only transactions an important performance issue in these applications. Although read-only transactions can be processed with conventional transaction processing algorithms, in many cases it is more efficient to process read-only transactions with special algorithms that take advantage of the knowledge that the transaction only reads (Lee et al., 1999; Pitoura and Chrysanthis, 1999). Concurrency control techniques for broadcast environments that exploit the semantics of read-only transactions are proposed in (Acharya et al., 1996; Lee et al., 1999; Pitoura and Chrysanthis, 1999). Different from them, our protocol enforces a weaker correctness criterion called weak serializability.

3. Weak-serializability for broadcast environments

As following, we formally define and analyze weak serializability which was called weak consistency in (Arvola and Robert, 1985; Kin and Reboat, 1999).

For a given state $s$ of a data item, we use $\text{return}(s, a)$ to denote the output produced by operation $a$, and $\text{state}(s, a)$ to denote the state produced after the execution of $a$. Given a history $H$ of events relating to transactions in $T$, $H^{(x)}$ is the projection of the history containing the operation invocations on a data item $x$. $H^{(x)} = a_1 \prec a_2 \prec \cdots \prec a_n$, indicates both the order of execution of the operations, $(a_i \text{ precedes } a_{i+1})$, as well as the functional composition of operations. Thus, a state $s$ of a data item produced by a sequence of operations equals the state produced by applying the history $H^{(x)}$ corresponding to the sequence of operations on the data item’s initial state $s_0(s = \text{state}(s_0, H^{(x)}))$. For brevity, we will use $H^{(x)}$ to denote the state of a data item produced by $H^{(x)}$, implicitly assuming initial state $s_0$ (Ramamritham and Calton, 1995). In addition, we use $H^T$ to denote the projection of the history containing the operations belonging to the transactions in a transaction set $T$, while $T_i \subseteq T$. Furthermore, $D(t)$ is used to denote the set of data items operated by $t$, $D_n(t)$ is used to denote the set of data items written by $t$, $D_r(t)$ is used to denote the set of data items read by $t$, and DB is used to denote the database.

**Definition 1** (Ramamritham and Calton, 1995). Two operations $a$ and $b$ conflict in a state produced by $H^{(x)}$, denoted by $\text{conflict}(H^{(x)}, a, b)$, if

$$(\text{state}(H^{(x)} \prec a, b) \neq \text{state}(H^{(x)} \prec b, a)) \lor \text{return}(H^{(x)}, b) \neq \text{return}(H^{(x)} \prec b, a)) \lor \text{return}(H^{(x)}, a) \neq \text{return}(H^{(x)} \prec b, a)).$$

Let $a_i[x]$ denote operation $a$ invoked by $t_i$ on data item $x$. $(a_i[x] \rightarrow b_j[x])$ implies that $a_i[x]$ appears before $b_j[x]$ in $H$.

**Definition 2** (Ramamritham and Calton, 1995). Let $t_i$ and $t_j$ be transactions $\in T$. Given a history $H$ of events relating to transactions in $T$, $\text{CSR}_H$, a binary relation on $T$, is defined as follows:

$$(t_i, \text{CSR}_H t_j), t_i \neq t_j \text{ iff } \exists x \exists a, b(\text{conflict}(H^{(x)}, a_i[x], b_j[x]) \land (a_i[x] \rightarrow b_j[x])).$$

Let $\text{CSR}_H^*$ be the transitive-closure of $\text{CSR}_H$; i.e.,

$$(t_i, \text{CSR}_H^* t_j) \text{ iff } [(t_i \text{CSR}_H t_j) \lor \exists t_k(t_i, \text{CSR}_H t_k \land t_k \text{CSR}_H^* t_j)].$$

$H$ is (conflict) serializable iff $\forall t \in T_\neg(t \text{CSR}_H^* t)$. To illustrate the practical implications of this definition, let us consider the case where all operations perform in-place updates. In this case, if transactions $t_i$ and $t_j$ have a $\text{CSR}_H$ relationship, i.e., $t_i$ has invoked an operation which conflicts with a previous operation by $t_j$, as long as $t_i$ is serialized before $t_j$, the conflict can be tolerated. Consider the precedence graph corresponding to the $\text{CSR}_H$ relation induced by a history $H$, denoted by...
The above definition states that the history $H$ is serializable, iff there are no cycles in the precedence graph $S_H$; i.e., the serialization order is acyclic (Ramamritham and Calton, 1995).

**Definition 3.** Let $t_i \in T$ and $t_j \in T$. Given a history $H$ of events relating to transactions in $T$, $R_H$, a binary relation associated with $H$, is defined as follows:

$$(t_iR_Ht_j), \quad t_i \neq t_j \text{ iff } \exists x (H, t_i \text{ reads the value of variable } x \text{ that was written by } t_j \text{ just now}).$$

Let $R^*_H$ be the transitive-closure of $R_H$; i.e.,

$$(t_iR^*_Ht_j) \text{ if } [(t_iR_Ht_k) \lor \exists t_k \in R_H^*(t_iR_Ht_j)].$$

**Definition 4.** Let $t$ be a read-only transaction that executes in a history $H$. Then, the set of associated transactions of $t$ in the history $H$, $A_H(t)$, is the minimal set closed under the following two rules:

(a) $t \in A_H(t)$;
(b) $\forall t_j \in T_S(\exists t_i \in T_S(t_i \in A_H(t) \land (t_iR_Ht_j)) \Rightarrow t_j \in A_H(t)).$

**Definition 5.** Suppose a transaction set $T$ and $T = T_u \cup T_r, T_u$ is the set of update transactions while $T_r$ is the set of read-only transactions. The history $H$ of events relating to transactions in $T$ is weak serializable iff

$$[(\forall t_i \in T_u \neg (t_iCSR^*_Ht_j)) \land (\forall t_j \in T_r \in A_H(t_j) \neg (t_iCSR^*_Ht_j)].$$

**Theorem 1.** Weak serializability is more relaxing than serializability.

**Proof.** We known that a history $H$ is serializable iff $S_H$ is acyclic. On the other hand, if $S_H$ is acyclic, it directly follows from Definition 5 that $H$ is weak serializable; besides, we can proof that $H$ may be weak serializable although $S_H$ has cycles. Consider the following history $H'$:

$$r_1(x) \prec r_2(y) \prec w_3(y) \prec w_2(x) \prec c_2 \prec r_1(y) \prec w_1(y) \prec c_1 \prec r_1(y) \prec c_1.$$ $S_{H'}$ shown in Fig. 1 has a cycle, but from Definition 5, we know that $H'$ is weak serializable. In summary, all of serializable histories are weak serializable, while there exists at least a weak serializable history ($H'$, for example) that is not serializable. Hence, we can conclude that weak serializability is more relaxing than serializability. □

![Fig. 1. Precedence graph of $H'$.](image)

**Theorem 2.** Weak serializability allows more concurrency control executions than BCC-TI does.

**Proof.** Suppose a transaction set $T$ and $T = T_u \cup T_r, T_u$ is the set of update transactions while $T_r$ is the set of read-only transactions. We known that BCC-TI allows a history $H$ iff $\forall r \in T_S$ $S_{H^0}^{H^0}$ is acyclic. On the other hand, if $S_{H^0}^{H^0}$ is acyclic, it directly follows from Definition 5 that $H$ is weak serializable; besides, we can proof that $H$ may be weak serializable although $S_{H^0}^{H^0}$ has cycles. Consider the history $H'$, too. $S_{H^0}^{H^0}$ shown in Fig. 1 has a cycle, but from Definition 5, we know that $H'$ is weak serializable. In summary, all of the histories that BCC-TI allows are weak serializable, while there exists at least a weak serializable history ($H'$, for example) that is not allowed by BCC-TI. Hence, we can conclude that weak serializability allows more concurrency control executions than BCC-TI does. □

**Definition 6.** An associated-precedence graph of a read-only transaction $t$ that executes in a history $H$, denoted by $G_H^t = (E, V)$, is a directed graph, where:

$$V = A_H(t);$$
$$E = E_1 \cup E_2;$$
$$E_1 = \{((t_i, t_j)| t_i \in V \land \exists x (r_i[x] \rightarrow w_j[x])\}$$
$$E_2 = \{((t_i, t_j)| t_i \in V \land \exists x (r_i[x] \rightarrow w_j[x])\};$$

**Theorem 3.** Suppose a transaction set $T$ whose events are recorded in history $H$, and $T = T_u \cup T_r, T_u$ is the set of update transactions while $T_r$ is the set of read-only transactions. Let $\forall t_i \in T_u \neg (t_iCSR^*_Ht_j), \forall t_j \in T_r S_{H^0}^{H^0}t_j$ is acyclic iff $G_H^t$ is acyclic.

**Proof (Only if).** From Definition 2 and 6, we know $\forall t_i \in T_c, G_H^t$ is a sub graph of $S_{H^0}^{H^0}$. Intuitively, if $S_{H^0}^{H^0}$ is acyclic, then $G_H^t$ is acyclic.

If (proof by contradiction). Assume $\exists t_i \in T_c, G_H^t$ is acyclic while $S_{H^0}^{H^0}$ has a cycle. Now there are two possibilities. Either $t_i$ is in the cycle, or not. From Definitions 3 and 4, we know $\forall t_k \in A_H(t_i) \land R_Ht_k$. This implies that $\forall t_k \in A_H(t_i)$ there is a path from $t_k$ to $t_i$ in $G_H^t$. Because $G_H^t$ is acyclic, $\forall t_k \in A_H(t_i)$ there is no edge from $t_k$ to $t_i$ in $G_H^t$. Combined with $t_i \in T_c$, it follows that the out-degree of $t_i$ in $S_{H^0}^{H^0}$ is zero. This contradicts the former case. Consider the latter case, from Definition 4 we know that there is no read-only transaction other than $t_i$ in $A_H(t_i)$. Then the cycle in $S_{H^0}^{H^0}$ must only have transactions in $T_u$. Combined with Definitions 2-4, it
follows that the cycle is also in $S_{HT_u}$. This contradicts $\forall t_i \in T_u \neg(t_i \text{CSR}_r_{HT_u} h_i)$ that implies $S_{HT_u}$ is acyclic. \hfill $\Box$

**Theorem 4.** Suppose a transaction set $T$ and $T = T_u \cup T_r$. $T_u$ is the set of update transactions while $T_r$ is the set of read-only transactions, a history $H$ of events relating to transactions in $T$ is weak serializable iff $\forall t_i \in T_u \neg(t_i \text{CSR}_r_{HT_u} t_i) \wedge (\forall t_j \in T_r G_{HT_r}$ is acyclic).

**Proof.** Follows from Definition 5 and Theorem 3. \hfill $\Box$

**Definition 7.** The value of a data item $d$ at time $\tau$ is called the state of $d$ at $\tau$, written as $S_\tau(d)$, i.e. $\forall d \in \text{DB}(S_\tau(d)) = V_\tau(d)$.

**Definition 8.** In a database, the set of all data items' states at $\tau$ is called the database state at $\tau$, written as $S_\tau(\text{DB}) = \{S_\tau(d)|d \in \text{DB}\}$.

**Definition 9.** In a database, the set of some data items’ states at time $\tau$ is called a database sub state at $\tau$, it is a subset of the database state at $\tau$, written as $S_\tau(A) = \{S_\tau(d)|d \in A, A \subseteq \text{DB}\}$.

**Theorem 5.** Suppose a transaction set $T$ and $T = T_u \cup T_r$. $T_u$ is the set of update transactions while $T_r$ is the set of read-only transactions. A read-only transaction $r (r \in T_r)$ has read some inconsistent state if and only if there exists a cycle in its associated-precedence graph.

**Proof (If).** Assume that there was a cycle as shown in Fig. 2 in the associated-precedence graph.

According to the above definitions, it follows that:

$S_0(\text{DB} - D_W(t_1)) = S_{t_1}(\text{DB} - D_W(t_1))$;

$S_0(D_W(t_1)) \neq S_{t_1}(D_W(t_1))$;

$S_{t_1}(\text{DB} - D_W(t_2)) = S_{t_2}(\text{DB} - D_W(t_2))$;

$S_{t_2}(D_W(t_2)) \neq S_{t_2}(D_W(t_2))$;

$\ldots$

$S_{t_{a-1}}(\text{DB} - D_W(t_a)) = S_{t_a}(\text{DB} - D_W(t_a))$;

$S_{t_{a-1}}(D_W(t_a)) \neq S_{t_a}(D_W(t_a))$.

Hence,

$S_0\left(\text{DB} - \bigcup_{i=1, a} D_W(t_i)\right) = S_{t_a}\left(\text{DB} - \bigcup_{i=1, a} D_W(t_i)\right)$;

$S_0\left(\bigcup_{i=1, a} D_W(t_i)\right) \neq S_{t_a}\left(\bigcup_{i=1, a} D_W(t_i)\right)$.

(1)

The non-equation (1) shows that the read-only transaction has read some inconsistent states.

**Only if.** There are two possibilities:

1. The read-only transaction $r$ read some data items updated, but no update transactions would update the data items read by the read-only transaction, therefore $D_r(t) \subseteq S_{t_a}(\text{DB})$ where $D_r(t)$ is the data set read by transaction $r$, and the read-only transaction read the consistent state after the execution of all dependent transactions.

2. A transaction updates some data items read by the read-only transaction, but the read-only transaction would only read non-updated items rather than modified ones, therefore $D_r(t) \subseteq S_{t_a}(\text{DB})$ and the read-only transaction read the consistent state before the execution of the update transaction. \hfill $\Box$

**Theorem 6.** Weak serializability is in polynomial time.

**Proof.** Follows from Definition 2, traditional serializability theory (Abraham et al., 1997; Ramamritham and Calton, 1995) and Theorem 4. \hfill $\Box$

For weak serializable, all of the update transactions must be serializable from each other while read-only transactions need not. However, every read-only transaction would still read consistent state of the database. In other words, weak serializability can maintain consistency of the database and the values read by read-only transactions, although read-only transactions need not serialize with each other. It has great significance for real-time broadcast systems to eliminate the communication load (especially upstream), in order to help more mobile real-time transactions to meet their deadlines and improve response time while maintaining consistency.

4. The protocol BCC-WSR

It is assumed that all update transactions are at the broadcast server, while all mobile transactions are read-only. Every mobile transaction has a soft deadline and is processed until it is committed, even the deadline is missed (Lee et al., 1999).

4.1. The server algorithm

For an update transaction $t$, the following procedure is performed:

\{While ($t$ is committing) Do

\{Broadcast the message that $t$ is committed with $D_w(t)$ together;\} \}
4.2. The mobile transaction algorithm

For a mobile real-time transaction \( r \), the following procedure is performed:

\[
\text{Repeat}
\]
\[
\text{Listen to the server, and filter the required data items and commit information;}
\]
\[
\text{While (A message that } t(\in T_u) \text{ is committed is received) Do}
\]
\[
\{\text{If } (\exists t' \in T_u \ (D_w(t) \cap (R(r) \cup D(t'))) \neq \Phi) \}
\]
\[
\text{Then } \{T_u = T_u \cup \{t\};}
\]
\[
\text{If (A cycle is constituted in the associated-precedence graph) }
\]
\[
\text{Then Abort and restart } r;\}
\]
\[
\text{Until } (R(r) = D(r));
\]

While \( R(r) \) is the set of data items that have been read by the mobile transaction \( r \), \( T_u \) is the set of update transactions that have been in the associated-precedence graph.

5. Performance evaluation

Among the few concurrency control techniques for transactional clients in broadcast environments, BCC-TI (Lee et al., 1999) has been shown to be more efficient than others for meeting transaction deadlines and improving response time. Our simulation experiments are aimed at comparing the performance of BCC-WSR and the method BCC-TI for concurrency control of mobile read-only transactions in broadcast disk environments. In this paper, we do not consider the influence of caching and disconnection on BCC-WSR that is going to be presented in detail in another paper.

5.1. Experimental setup

Our performance model is similar to the ones presented in (Acharya et al., 1995; Kayan and Ulusoy, 1999; Pitoura and Chrysanthis, 1999). For simplicity, we assume a flat broadcast disk with one server and one client. Table 1 lists the simulation parameters. The simulation runs in unit of bit-time that is the time to transmit a single bit. The server periodically broadcasts a total of Server-Objects data items, while the client accesses only a subset ClientObjects of the objects that the server broadcasts. Data items of HotObjects are very frequently updated with probability HotProbUpdate on the server while ClientHotObjects hot objects are frequently read with probability HotProbReading from the mobile client. Each object is of size ObjectSize bits. An update transaction arrives at the server every UpdateArrive Kbit-times while A read-only transaction arrives every QueryArrive Kbit-times. The client waits ThinkTime Kbit-times and then makes the next read request. The deadline is (current time + slack factor * predicted execution time), where slack factor is uniformly distributed in \([\text{MinSlack, MaxSlack}]\), and predicted execution time is a function of transaction length, mean inter-operation delay, mean inter-transaction delay, broadcast cycle length. These protocols are compared in the average response time and percentage of read-only transactions that missed deadlines. For each configuration of each experiment, the final results were evaluated as averages over 20 independent runs. Each configuration was executed for 500 transactions originating at each site. Ninety per cent confidence intervals were obtained for the performance results. The width of the confidence interval of each data point is within 4% of the point estimate.

5.2. Experimental results and discussions

BCC-TI enforces local serializability, while BCC-WSR enforces weak serializability. Weak serializability can help mobile transactions to meet their deadlines and improve response time because: (1) Weak serializability that is more relaxing than serializability can reduce number of aborts. (2) It can eliminate the communication load (especially upstream) for data conflict detection.

Generally, the increase in the update transaction arrival rate increases the probability of data conflicts. More the probability of data conflicts, more aborted read-only transactions. Hence, longer average response time, and higher miss rate. The performance of the two protocols in miss rate is shown in Fig. 3. It is depicted that as updates get more frequent the miss rate increases more rapidly for BCC-TI than for BCC-WSR. Fig. 4 shows that the average response times in both protocols increase, while the average response time in BCC-WSR increases slowly than BCC-TI. Compared to BCC-TI, the improvement aroused by BCC-WSR in average response time is about 22%.
Figs. 5 and 6 show the impact of update transaction length on the two protocols in terms of miss rate and average response time. Longer update transactions result in more updates at the server in every broadcast cycle. Hence, more data conflicts would be caused. Therefore, the miss rates and the average response times increase with update transaction length for both protocols. However, the miss rate and the average response time for BCC-WSR increase much more slowly than BCC-TI.

Figs. 7 and 8 depict the relative performance in percentage of read-only transactions that missed deadlines and the average response time of both protocols as functions of mobile transaction length. When the length of mobile transactions is short, the performance of BCC-TI and BCC-WSR are similar, with BCC-WSR having a slightly better performance. The differences in their performance become more significant when the length of mobile transactions become longer. The average response time and the percentage of read-only transactions that missed deadlines for BCC-TI increase sharply after the length of mobile transactions is 7, indicating that the impact of data conflicts is more significant when the length of mobile transactions has been longer.

Longer the size of objects, longer the length of a broadcast cycle, and longer the average waiting time for a read. The prolonged waiting time will increase the response time of the read-only transaction and the probability of data conflict with update transactions. Fig. 9 gives the miss rate of the read-only transactions issued by mobile transactions as a function of the size of object.
objects, while Fig. 10 shows the average response time of both protocols as a function of data object size. As the size of object increases, the miss rates and the average response times of both protocols increase. However, BCC-TI is more sensitive than BCC-WSR because of more aborts and restarts involved.

In Figs. 11 and 12, the impact of the number of objects in the database on miss rate and average response time is depicted. With the increasing number of objects in the database, the BCC-WSR is still better. Although the probability of accessing an object decreases with the increasing number of objects, the length of the broadcast cycle increases. Consequently, it increases the number of update transactions that executed in every broadcast cycle. Hence, the number of possible conflicts would be increased. Therefore, the miss rates and the average response times of both protocols increase with the number of objects in the database. As depicted in Figs. 11 and 12, the miss rate and the average response time for BCC-WSR increase more slowly than BCC-TI. The advantage of BCC-WSR is demonstrated once again.

6. Conclusions

Considering the unique characteristics of real-time broadcast environments, serializability is too strong as a correctness criterion and not suitable for mobile real-time transactions, while relaxing serializability such as epsilon and similarity serializability may sacrifice database consistency to some extent. On the contrary, weak serializability that is more relaxing than serializability and maintains database consistency appears proper in real-time broadcast environments. Furthermore, BCC-WSR based on weak serializability outperforms BCC-TI in all the experiments. We are now interested in the issues of caching and disconnection treatment in real-time broadcast environments.

References


Dang Depeng, born in 1970. He received his B.S. and M.S. degrees from Xidian University and Hubei University in 1991 and 1999, respectively. Now he is a Ph.D. candidate in Huazhong University of Science and Technology. His research interests include database performance issues, mobile and real-time computing.

Liu Yunsheng, born in 1940. He received his B.S. degree from Wuhan University in 1964. Now he is a professor, Ph.D. supervisor in the Huazhong University of Science and Technology. His research interests include advanced database and information systems, software methodologies and technologies, and real-time and mobile computing. Prof. Liu is senior member of the IEEE Computer Society and the ACM.