http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (1): 83-87, 2006
ISSN 1812-5638

A Synoptic Review of Reconciliation Algorithm for Mobile Databases

[1]Mohammod Shamim Hossain and [2]M. Anwar Hossain
[1]School of Science and Technology, Bangladesh Open University, Board Bazar, Gazipur-1705, Bangladesh
[2]ITsoft International, Dhaka-1230, Bangladesh

Abstract: The explosive growth of mobile communications has opened the possibility of extending traditional database management functionality to the mobile computing environment. Databases running in the mobile environment have the ability to allow optimistic replication and local updates on the disconnection clients. Reconciliation of conflicting updates is a fundamental problem in mobile systems. This study presents a multi version reconciliation algorithm to resolve this issue and discusses the benefits and limitations of such scheme. One of the benefits to avoid cascading abort has been appreciated while some limitations in terms of symmetric reconciliation and concurrence control have been criticized and explored further.

Key words: Mobile database, concurrence control, multi version, time stamp, reconciliation

INTRODUCTION

With the rapidly expanding cellular, wireless and satellite communications, it is possible for mobile users to access any data, anywhere, at any time. Mobile database is the solution to empower the mobile users. But due to higher cost and difficulty in maintaining persistent communication, a mobile database needs to work under disconnection. At that time mobile units cannot access any shared data. Optimistic replication approaches have been proposed to handle this disconnection problem[1], where the mobile units can locally replicate and operate on the shared data. On reconnection, the local updates can be transmitted to the central database server, which might eventually conflict with other updates. Phatak and Badrinath[2] presented a scheme to detect and resolve such conflicts in order to prevent any database inconsistencies.

The approach considers an extended client server architecture, which can be realized by observing Fig. 1. The host/central database server contains the primary copies of all data items while the mobile clients have the replica of the server database. When there is active connection between client and server, the client transactions are serialized and committed on the server globally. But when there is no connection between them, the client transactions commit locally against its replica. These committed local transactions are available to other local transactions as well. To upload the work performed by the mobile database users, all local transactions are sent to the server on regaining the connection. The transactions will be committed globally if they pass the serializabilty test. Otherwise the transaction will end without commit, which refers to irresolvable conflicting transaction. This process of testing serializabilty, conflict resolution and serialization is regarded as reconciliation.

To address the reconciliation problem in a more generic way, multi version approach has been combined with the single version optimistic concurrence control[3]. As in a multi version scheme, the system keeps several versions x_1, x_2, …, x_k of each data item x and provides snapshot isolation[4], client transactions can be reconciled

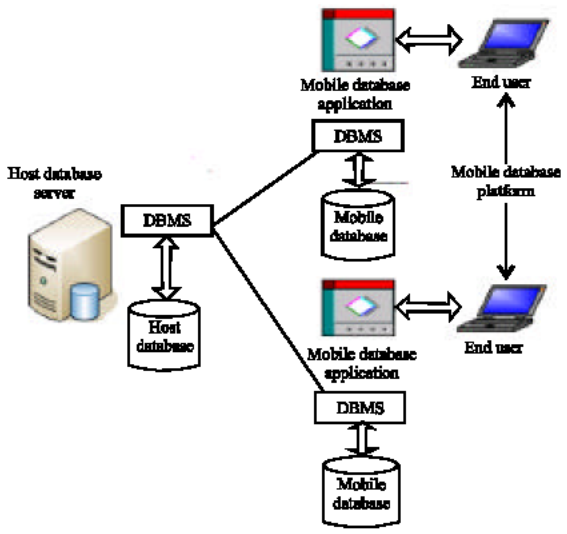

Fig. 1: Typical client-server mobile environment

Corresponding Author: Mohammod Shamim Hossain, School of Science and Technology, Bangladesh Open University, Board Bazar, Gazipur-1705, Bangladesh

against any of these old version. With snapshot isolation, multi version transaction can commit if and only if data items in its write set were not overwritten after this transaction read them.

There are several literatures[5-9] related to optimistic replication scheme of mobile databases. One of such scheme proposed by Gray *et al.*[10] to guarantee data consistency is to provide individual applications with a view of the database that is consistent with their own actions. This is known as two-tier (one tier is the mobile disconnected nodes and the other tier is the connected base nodes) replication algorithm, which allows mobile applications to propose tentative transactions at the mobile client. These tentative transactions are performed over data replicas stored at the client while the clients are disconnected. When the mobile client reconnects, the tentative updates are reprocessed as transactions on the server that stores the master copy of the data. The tentative transactions may fail due to conflicts with other transactions. The mobile client who originated the transaction is then informed about the failure and why it happened. However, this technique renders systems unscalable and may result in an unacceptable number of failed transactions after the clients reconnect.

Another scheme known as the Bayou architecture[11] support shared databases that can be read and updated by disconnected users. This Bayou system detects update conflicts in an application specific manner. The system, along with detecting updates conflicts, also provides means for resolving such conflict. On connection to the data repositories, updates are reconciled. This reconciliation takes place by aborting all updates made by both connected servers and then propagating both sets of updates in time stamp order. However, Bayou requires specialized knowledge to provide conflict resolution. This study gives some solution of reconciliation, which does not require specialized knowledge to provide conflict resolution. Some other articles[5-7] which discusses about multiversion concurrency control and reconciliation

MULTIVERSION RECONCILIATION ALGORITHM

The algorithm[2] has been implemented in order to reconcile the transactions of the disconnected clients with the central database server when connections are reestablished. The algorithm proceeds by considering each client transaction in turn and progressively computing snapshots and attempting to serialize client transactions against these snapshots. To achieve these two goals: least cost reconciliation and snapshot isolation, two snapshot functions are required. They are:

- Backward/normal snapshot function $S\downarrow(v)$: used to determine the input snapshot of the transaction.
- Forward or reverse snapshot function $S\uparrow(v)$: used to determine whether the transaction's updates can be serialized against the backward snapshot.

Here the database is considered as a collection of data items along with their versions. The data items are drawn from a universe X. The database at any given instant exists in a particular state, D. To get an overview of the algorithm, following definitions[2] are required to consider:

Definition 1: Database State and Value Function.
The state of the database on the server at any instant of time consists of a set of elements $D \subset X \times \check{Z}^+$, a collection of a finite number of versions of data items drawn from X and a value: $D \rightarrow U_{x \in X}$ domain (x) function that maps data items and versions to the actual values of the data items.

Definition 2: Version snapshot function.
A Version snapshot function $S: \check{Z}^+ \rightarrow D$, maps a non-negative integral timestamp into a snapshot of the database. For each timestamp value it yields a collection of versions of data items with the following properties:

- Every snapshot S(v) is a subset of the database state.
- For any given snapshot S(v), if the data item was written by transaction with timestamp v, then version v of the data item is in S(v).
- No data item in S(v) can have version number greater than v.
- Only one version of a data item can be present in a snapshot.
- A snapshot S(v) has the latest versions of the data items that are less than or equal to v. Note that if a data item x was introduced into the database by a transaction with timestamp greater than v, then x cannot be in S(v).

Definition 3: Read, Write and Read-Write Sets.
Let us define three partial value snapshots for each T:

- $RSET^v(T)$: consists of all data items that were read but not written by T.
- $RWSET^v(T)$: consists of all data items that were read and then written by the transaction.
- $WSET^v(T)$: consists of all the data items that are written by T before they are read (T blind writes)
- $READSET^v(T) = RSET^v(T) \cup RWSET^v(T)$
  $WRITESET^v(T) = RWSET^v(T) \cup WSET^v(T)$

Conflict resolution: the client optionally defines Conflict Resolution function CR_{Tc} for each client transaction T_c. The CR_{Tc} is provided whenever client seeks reconciliation of a transaction. If CR_{Tc} is defined, the client must also define a cost function C_{Tc}. The conflict resolution and cost functions are defined as follows:

- CR_{Tc} takes as input three value snapshots: the READSET consisting of data items and values read by T_c, the WRITESET and values generated by T_c and a new values snapshot against which T_c needs to be serialized.
  CR_{Tc} ($READSET^V(T_c)$, $WRITESET^V(T_c)$, S^v_{in}) = $NEWWRITESET^V(T_c)$
- C_{Tc} takes the same inputs as CR_{Tc} and returns an integral cost value that indicates the cost of resolving conflicts for that set of inputs.
  C_{Tc}($READSET^V(T_c)$, $WRITESET^V(T_c)$, S^v_{in}) = $|READSET^V(T_c) - SREADSET^V(T_c)|$,

where, $SREADSET^V(T_c) = \{ <x, i> | <x, i> \in S^v_{in} \wedge \exists i' : <x, i'> \in READSET^V(T_c)\}$. If the client does not specify these functions, the server uses defaults defined as follows:

Definition 4: Default Cost Function.

$$C_{TC}(READSET^V(T_c), WRITESET^V(T_c), S^v_{in}) = \begin{cases} 0 & \text{if } READSET^v(T_c) \subseteq S^v_{in} \\ \infty & \text{Otherwise} \end{cases}$$

Definition 5: Default Conflict Resolution Function.

$$CR_{Tc}(READSET^V(T_c), WRITESET^V(T_c), S^v_{in}) = \begin{cases} WRITESET^V(T_c), & \text{if } READSET^v(T_c) \subseteq S^v_{in} \\ \text{Undefined} & \text{Otherwise} \end{cases}$$

Now the two algorithms based on Phatak[1] are presented using a very high level notation.

Algorithm 1: Reconciliation algorithm

- Inputs: DB state D, Transaction Tc, CR_{Tc}, C_{Tc}
- Take a client transaction; compute a snapshot by considering each timestamp and check if it has least cost using C_{Tc}
- Create new writeset using CR_{Tc}
- Ensure all writes in the new writeset are serialized after the latest version of the data items and ensure there is no blind write.
- If the transaction cannot be serialized, abort it. Otherwise reconcile.
- Output: Change in DB state if successful. Else no changes in DB state.

Algorithm 2: Reintegration algorithm

- Inputs: All transactions in the client (C), DB state D, CR_{Tc}, C_{Tc}
- Take by turn, each Tc from all transaction lists
  - Call Algorithm 1 (D, Tc, CR_{Tc}, C_{Tc})
- Output: Change in DB state if any Tc succeeds. Else no changes in DB state.

In a transaction-centric environment, the system should disallow Dirty Write, Dirty Read, Phantom Phenomena, Phantom Anomaly, Read Skew and Lost Update in order to provide snapshot isolation[4].

As the algorithm operates on committed data on the server, it does not allow Dirty Read or Fuzzy Read. In this system, reconciliation is considered as a second instance of read predicate. So Phantom Phenomena or Phantom anomaly will not occur. The client transaction gets reconciled against a snapshot of the database, so anomaly Read Skew cannot occur.

Uncommitted updates do not appear as a part of database's state and the writes of client transaction will be serialized after committed writes, Dirty Write will no occur.

PROS AND CONS OF THE ALGORITHM

One of the benefits of this algorithm is that it prevents cascading abort. That means even when one client transaction cannot be committed on the server and aborts, other client transactions will not be aborted automatically and can still try to commit on the server.

Another benefit of this algorithm is that it does not require application specific specialized knowledge for the reconciliation process.

A limitation of the proposed multiversion reconciliation algorithm is that it does not provide Symmetric Reconciliation. It only works in a client server mode. This means one of the hosts in the mobile network must act as a central database server, which will hold globally committed versions of all the data items. We have addressed this issue in order to remove this limitation in the subsequent section.

Another limitation of this algorithm is that it does not support robust concurrency control mechamism. Because it requires each client transaction be reconciled by an atomic operation on the server. However, this will consume much time, as multiple snapshots need to be computed and tested thereby affecting system

throughput. The Authors tried to address this issue by providing a preliminary technique, which is not solid in its domain.

DISCUSSION

Here the benefit of the algorithm regarding preventing cascading abort of client transactions is elaborated.

In the algorithm, the responsibility of conflict resolution has been assigned to the client. The client performs this duty by providing the conflict resolution (CR_{Tc}) and cost functions (C_{Tc}) for each of its transaction. It should be noted that in the absence of any application specified conflict resolution, this algorithm works at least as well as standard optimistic concurrency control adapted to disconnected systems. In the single version case, the algorithm reduces to optimistic concurrency control in the absence of explicit conflict resolution. In this case the algorithm is a true generalization of standard optimistic techniques. So, a application is guaranteed at least this level of performance. Of course, use of multiple versions with snapshot isolation and conflict resolution further enhances performance.

In the absence of these functions, server provides default functions. Tc here is denoted as current transaction. This transaction will rarely be aborted as the conflict resolution and cost functions cover most of the aspect. Moreover the transaction is made serializable against the latest snapshot of committed versions. This latest snapshot is taken based on a timestamp.

When timestamp-based techniques are used for concurrency control on the server and client transaction tries to commit on the current snapshot, its write might conflict with other active transactions. In this situation, the client transaction Tc might be aborted. But for the server to maintain multiversioning, other transactions that read data from Tc (dirty read, as it is aborted) will not be aborted.

Let's see a sample history to show the proof of the above fact:

Server	Transaction	Client	Transaction
$w_0[x_0]=1$, c_0,	T_0	$x_0=1$ (downloaded)	
$r_1[x_0]=1$, $w_1[x_1]=2$, c_1,	T_1	$r_1'[x]=1$, $w_1'[x]=3$, c_1',	T_1'
$r_2[x_1]=2$, $w_2[x_2]=3$, c_2	T_2	$r_2'[x]=3$, c_2'	T_2'

As can be seen in this example history, the client is following single version scheme. If client does not provide explicit conflict resolution function, the default conflict resolution function of the server produces the new writeset (value snapshot) according to algorithm as equals to {<x, 3>}. Transaction T_1' can only be serialized with T_1 on the server and hence will be aborted on reconciliation with the server.

If we do not consider any intertransactional dependencies, even T_1' is aborted, T_2' can be serialized with timestamp 3-Δ giving the following serial history:

Server: $w_0[x_0]=1, c_0, r_1[x_0]=1, w_1[x_1]=2, c_1, r_2[x_1]=2, w_2[x_2]=3, c_2, r_2'[x_3]=3, c_2'$

It should be noted, that the above scenario occurs as T_2' sees the same value snapshot on the server that it sees on the client. In this way, cascaded aborts are prevented on the client.

- Now we explain one of the limitations of the algorithm in terms of Symmetric Reconciliation. To address this issue, let us consider two disconnected clients of a server. They are independently running transactions on two replicas of the server. Here the two clients are connected with each other. Now if two clients wish to mutually synchronize their local replicas without connecting to the server, how will they do that? In the proposed algorithm there is no way to handle this situation.

To illustrate this, we shall assume that a global timestamp scheme is available for ordering events between the two clients that is consistent with the global commit timestamps on the server and the order of local commits on each client.

Further, suppose that the two replicas are offline during the reconciliation process. Let R_1 be the replica on the first client and R_2 be the replica on the second client. Furthermore, let R_{11} and R_{21} be the corresponding replicas with the effects of all local transactions undone. Then by their definition of state of the database, $R_{11} \cup R_{21}$ is a subset of global database state (D). This is because both R_{11} and R_{21} are exactly what the clients must have originally replicated from the server. Hence both R_{11} and R_{21} by definition must be subsets of global database state. Since a union of subsets of a set must also be a subset of the set, $R_{11} \cup R_{21}$ must be a subset of global database state.

Now, we can run the reintegration algorithm with the database state D set to R_{11} U R_{21} and the union of sets of local transactions from both clients. The algorithm can be run on either client, or first on one client with its set of local transactions and then on the other client with the other set of local transactions. In either case, timestamp order will be determined by the global timestamp order on the clients.

CONCLUSIONS

The stated algorithm deals with the reconciliation of disconnected client transactions on the server. The algorithm is not unique in its identity, as it has adopted the ideas from several other researches[10,11]. But here no application specific specialized knowledge is required. The algorithm approached to solve the reconciliation problem by introducing conflict detection and conflict resolution. Conflict detection is the responsibility of the server while conflict resolution is the responsibility of the client. Several pros and cons of the algorithm have been explored in order to understand its implications. Preventing cascading abort is one of the benefits the algorithm provides. On the other hand there are some limitations with the algorithm. It does not provide symmetric reconciliation and robust concurrency control mechanism.

REFERENCES

1. Satyanarayanan, M., 1989. Coda: A highly available file system for a distributed workstation environment. Proceeding of IEEE Workshop on Workstation Operating System, pp: 447-459.
2. Phatak, S.H. and B.R. Badrinath, 1999. Multiversion reconciliation for mobile databases. ICDE., pp: 582-589.
3. Kung, H.T. and J.T. Robinson, 1981. On optimistic methods for concurrency control. ACM Trans. Database Sys., 6: 213-226.
4. Berenson, H., P. Bernstein, J. Gray, J. Melton, E. O'Neil and P. O'Neil, 1995. A critique of ANSI SQL isolation levels. Proceeding of ACM SIGMOD International Conference of Management Data, pp: 1-10.
5. Graham, P.C.J. and K.E. Barker, 1994. Effective Optimistic Concurrency Control in Multiversion Object Bases. Springer-Verlag LNCS-858, pp: 313-328.
6. Reza-Hadaegh, A., P.C.J. Graham and K.E. Barker, 1994. An architecture and model for processing transactions in multiversion object base systems. Proceeding of 2nd Mid-Continent Information and Database Systems
7. Burger, A., V. Kumar and M. L. Hines, 1997. Performance of multiversion and distributed two-phase locking concurrency control mechanisms in distributed databases. Inform. Sci., 96: 29-152.
8. Imelinski, T. and B.R. Badrinath, 1994. Mobile wireless computing: challenges in data management. ACM Commun., 37: 18-28.
9. Katz, R. and R. Weiss, 1984. Design of transaction mangement. Proceeding of 21st Design Automation Conference, pp: 692-693.
10. Gray, J., P. Helland, P.E. O'Neil and D. Shasha, 1996. The dangers of replication and a solution. Proceeding of ACM SIGMOD, pp: 173-182.
11. Demers, A., K. Petersen, M. Spreitzer, D. Terry, M. Theimer and B. Welch, 1994. The bayou architecture: Support for data sharing among mobile users. Proceeding of IEEE Workshop on Mobile Computing Systems and Applications, pp: 2-7.