Wireless Sensor Network, **2010, 2, 254-263**
doi:10.4236/wsn.2010.23035 Published Online March 2010 (http://www.scirp.org/journal/wsn)

ACTIVE-A Real Time Commit Protocol

Udai Shanker, Nikhil Agarwal, Shalabh Kumar Tiwari, Praphull Goel, Praveen Srivastava
Department of Computer Science and Engineering, M. M. M Engineering College, Gorakhpur, India
E-mail: {*udaigkp, nikhilmmmec, shalabhmmmec, goelpraphul, praveen*047}*@gmail.com*
Received November 19, 2009; *revised November* 30, 2009; *accepted December* 15, 2009

Abstract

Many existing real time commit protocols try to improve system performance by allowing a committing cohort to lend its data to an executing cohort, thus reducing data inaccessibility. They block the borrower from sending WORKDONE/PREPARED message and restrict them from lending data so that transaction abort chain is limited to one. Thus, transaction execution time increases. This paper proposes a modified real time commit protocol for distributed real time database systems (DRTDBS), Allow Commit Dependent and in Time borrowers for Incredible Value added data lending without extended abort chain (ACTIVE), where borrower cohorts are categorized as commit and abort dependent. Further, the commit dependent borrowers can lend data to executing cohorts with still limiting the transaction abort chain to one only and reducing the data inaccessibility. Also, an incoming executing cohort having borrowing factor greater than one can only borrow the dirty data items from lender. This minimizes the fruitless borrowing by the cohort. The performance of ACTIVE is compared with PROMPT, 2SC and SWIFT protocols for both main memory resident and disk resident databases with and without communication delay. Simulation results show that the proposed protocol improves the system performance up to 4% as transaction miss percentage.

Keywords: Distributed Real Time Database System, Commit Protocol, Conflict Resolution, Dependency, Lender, Borrower

1. Introduction

Maintenance of transaction's ACID semantics are the well known complexities that real time distributed database have to contend with when operating on the distributed data. Several important factors contribute to the difficulty in meeting transaction deadlines in DRTDBS. Data conflicts are one of the most important factors amongst the transactions. Two kinds of conflicts between transactions [1,2] arise. One occurs between executing transactions, which is resolved by a concurrency control protocol to ensure distributed transaction serializability; the other occurs between executing-committing transactions, which is resolved by a commit protocol to ensure distributed transaction atomicity. Limited work is done in case of executing-committing conflicts.

Database systems are currently being used as backbone to thousands of applications, which have very high demands for availability and fast real-time responses. A large part of the workload consists of read, write-blind and updates transactions against DRTDBS which these applications generate. Business transactions using these applications in the absence of real time could lead to financial devastations and in worst case cause injuries or deaths. Examples include telecommunication systems, trading systems, online gaming, sensor networks etc. Typically, a sensor network consists of a number of sensors (both wired and wireless) which report on the status of some real-world conditions. The conditions include sound, motion, temperature, pressure and moisture, velocity etc. The sensors send their data to a central system that makes decisions based both on current and past inputs. To enable the networks to make better decisions, both the number of sensors and the frequency of updates should be increased. Thus, sensor networks must be able to tolerate an increasing load. For applications such as Chemical Plant Control, Multi Point Fuel Injection System (MPFI), Video Conferencing, Missile Guidance System etc., data is needed in real-time, and must be extremely reliable and available as any unavailability or extra delay could result in heavy loss. Many applications listed above using DRTDBS require distributed transaction to be executed at more than one site. To maintain consistency, a commit protocol ensures that either all the effects of the transaction persist or none of them persist. Failure of site or communication link and loss of messages do not hamper the transaction processing. Commit

protocols must ensure that little overheads are laid upon transactions during processing. Hence, the need of developing a new commit protocol for better performance of DRTDBS arises.

The two phase commit protocol (2PC) referred to as the Presumed Nothing 2PC protocol (PrN) is the most commonly used protocol in the study of DDBS [3–5]. It ensures that sufficient information is force-written on the stable storage to reach a consistent global decision about the transaction. A number of 2PC variants [6] commit protocols have been proposed and can be classified into following four groups [7].

1) Presumed Abort/Presumed Commit
2) One Phase
3) Group Commit
4) Pre Commit/Optimistic

Presumed commit (PC) and presumed abort (PA) [8] are based on 2PC. Soparkar *et al*. [9] have proposed a protocol that allows individual site to unilaterally commit. Gupta *et al*. [10,11] proposed optimistic commit protocol and its variant. Enhancement has been made in PROMPT commit protocol [12], which allows executing transactions to borrow data in a controlled manner only from the healthy transactions in their commit phase. However, it does not consider the type of dependencies between two transactions. The impact of buffer space and admission control is also not studied. In case of sequential transaction execution model, the borrower is blocked for sending the WORKDONE message and the next cohort can not be activated at other site for its execution. It will be held up till the lender completes. If its sibling is activated at another site anyway, the cohort at this new site will not get the result of previous site because previous cohort has been blocked from sending the WORKDONE message due to being borrower [13]. In shadow PROMPT, a cohort forks off a replica of the transaction, called a shadow, without considering the type of dependency whenever it borrows a data page.

Lam *et al*. [1] proposed deadline-driven conflict resolution (DDCR) protocol which integrates concurrency control and transaction commitment protocol for firm real time transactions. DDCR resolves different transaction conflicts by maintaining three copies of each modified data item (before, after and further) according to the dependency relationship between the lock requester and the lock holder. This not only creates additional workload on the systems but also has priority inversion problem. The serializability of the schedule is ensured by checking the before set and the after set when a transaction wants to enter the decision phase. The protocol aims to reduce the impact of a committing transaction on the executing transaction which depends on it. The conflict resolution in DDCR is divided into two parts (a) resolving conflicts at the conflict time; and (b) reversing the commit dependency when a transaction, which depends on a committing transaction, wants to enter the decision phase and its deadline is approaching.

If data conflict occurs between the executing and committing transactions, system's performance will be affected. Pang Chung-leung and Lam K. Y. [2] proposed an enhancement in DDCR called the DDCR with similarity (DDCR-S) to resolve the executing- committing conflicts in DRTDBS with mixed requirements of criticality and consistency in transactions. In DDCR-S, conflicts involving transactions with looser consistency requirement and the notion of similarity are adopted so that a higher degree of concurrency can be achieved and at the same time the consistency requirements of the transactions can still be met. The simulation results show that the use of DDCR-S can significantly improve the overall system performance as compared with the original DDCR approach.

Based on PROMPT and DDCR protocols, B. Qin and Y. Liu [14] proposed double space commit (2SC) protocol. They analyzed and categorized all kind of dependencies that may occur due to data access conflicts between the transactions into two types commit dependency and abort dependency. The 2SC protocol allows a non-healthy transaction to lend its held data to the transactions in its commit dependency set. When the prepared transaction aborts, only the transactions in its abort dependency set are aborted and the transactions in its commit dependency set execute as normal. These two properties of the 2SC reduce the data inaccessibility and the priority inversion that is inherent in distributed real-time commit processing. 2SC protocol uses blind write model. Extensive simulation experiments have been performed to compare the performance of 2SC with that of other protocols such as PROMPT and DDCR. The simulation results show that 2SC has the best performance. Furthermore, it is easy to incorporate it in any commit protocol.

Ramamritham *et al*. [15] have given three common types of constraints for the execution history of concurrent transactions. The paper [16] extends the constraints and gives a fourth type of constraint. The weak commit dependency and abort dependency between transactions, because of data access conflicts, are analyzed. Based on the analysis, an optimistic commit protocol Two-Level Commit (2LC) is proposed, which is specially designed for the distributed real time domain. It allows transactions to optimistically access the locked data in a controlled manner, which reduces the data inaccessibility and priority inversion inherent and undesirable in DRTDBS. Furthermore, if the prepared transaction is aborted, the transactions in its weak commit dependency set will execute as normal according to 2LC. Extensive simulation experiments have been performed to compare the performance of 2LC with that of the base protocols PROMPT and DDCR. The simulation results show that 2LC is effective in reducing the number of missed transaction deadlines. Furthermore, it is easy to be incorporated with the existing concurrency control protocols.

Udai Shanker *et al*. [17] proposed SWIFT protocol. In SWIFT, the execution phase of a cohort is divided into two parts, locking phase and processing phase and then, in place of WORKDONE message, WORKSTARTED message is sent just before the start of processing phase of the cohort. Further, the borrower is allowed to send WORKSTARTED message, if it is only commit dependent on other cohorts instead of being blocked as opposed to PROMPT. This reduces the time needed for commit processing and is free from cascaded aborts. However, SWIFT commit protocol is beneficial only if the database is main memory resident. Based on the SWIFT protocol, Dependency Sensitive Shadow SWIFT (DSS-SWIFT) protocol [18] was proposed, where the cohort forks off a replica of itself called a shadow, whenever it borrows dirty value of a data item, and if, the created dependency is abort type as compared to creating shadow in all cases of dependency in Shadow PROMPT. Also the health factor of cohort is used for permitting to use dirty value of lender rather than health factor of transaction as whole.

Here, our modified work proposes a commit protocol ACTIVE, which allows lender to lend data to healthy borrowers. Also, commit dependent cohorts are further allowed to lend data. This protocol is beneficial both for main memory and disk resident databases.

The remainder of this paper is organized as follows. Section 2 introduces the distributed real time database system model. Section 3 discusses the type of dependencies created between conflicting cohorts. Section 4 states the condition for fruitful borrowing. Section 5 presents ACTIVE commit protocol and its pseudo code. Section 6 discusses the simulation results and Section 7 gives an outline for future research directions. Finally, Section 8 concludes the paper.

2. Distributed Real Time Database System Model

The common model for DRTDBS is given below in **Figure 1**. The structure of our simulation model including the description of its various components such as system model, database model, network model, cohort execution model, locking mechanism and the model assumptions is given below [17,18]. At each site, two types of transactions are generated: global transactions and local transactions. Each global transaction consists of m cohorts, where m is less than or equal to the number of database sites N_{site}. We use the same model for local and global transactions. Each local transaction has a coordinator and a single cohort both executing at the same site. Each transaction consists of N_{oper} number of database operations. Each operation requires locking of data items and then processing.

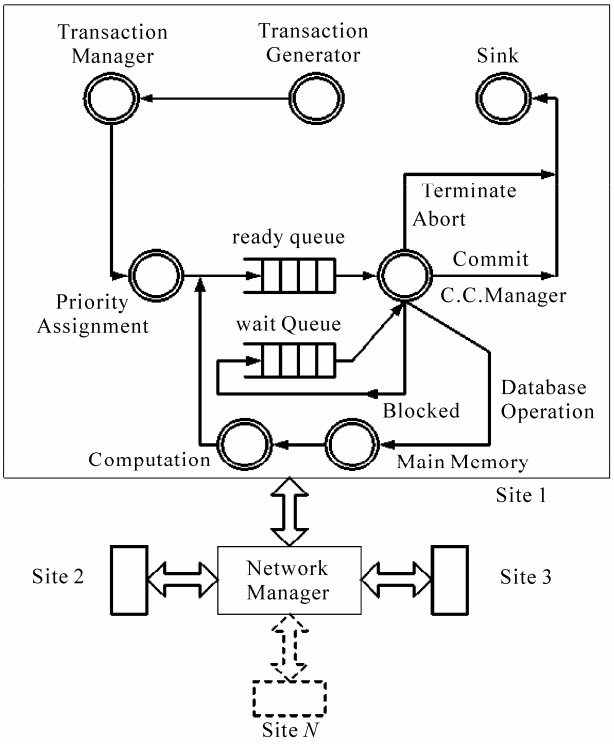

Figure 1. Distributed real-time database system model.

2.1. System Model

Each site consists of a transaction generator, a transaction manager, a concurrency controller, a CPU, a ready queue, a local database, a communication interface, a sink and a wait queue. The transaction generator is responsible for creating the transactions independent to the other sites using Poisson distribution with the given inter-arrival time. The transaction manager generates cohorts on remote site on behalf of the coordinator. Before a cohort performs any operation on a data item, it has to go through the concurrency controller to obtain a lock on that data item. If the request is denied, the cohort is placed in the wait queue. The waiting cohort is awakened when the requested lock is released and all other locks are available. After getting all locks, the cohort accesses the memory and performs computation on the data items. Finally, the cohort commits/aborts and releases all the locks that it is holding. The sink component of the model is responsible for gathering the statistics for the committed or terminated transactions.

2.2. Database Model

The database is modeled as a collection of data items that are uniformly distributed across all the sites. Transactions make requests for the data items and concurrency control is implemented at the data item level. No replication of data items at various sites is considered here.

2.3. Network Model

A communication network interconnects the sites. There is no global shared memory in the system. All sites communicate via messages exchange over the communication network. The network manager models the behavior of the communications network.

2.4. Cohort Execution Model

In our work, we have considered the parallel execution of cohorts. The coordinator of the transaction spawns all cohorts together by sending messages to remote sites to activate them, lists all operations to be executed at that site and then cohorts may start execution at the same time in parallel. The assumption here is that a cohort does not have to read from its sibling and operations performed by one cohort during its execution are independent of the results of the operations performed by other cohorts at some other sites. In other words, the sibling cohorts do not require any information from each other to share.

2.5. Locking Mechanism

The main technique used to control concurrent execution of transactions is based on the concept of locking data items. A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. Generally there is one lock for each data item in the database. Locks are means for synchronizing the access of concurrent transactions to the database items. A transaction is said to follow the two phase locking protocol if all locking operations precede the first unlock operation in the transaction. There is a number of variations of the two phase locking (2PL) such as static two phase locking (S2PL) and dynamic two phase locking (D2PL). The static 2PL (S2PL) requires a transaction to lock all needed data items before the transaction begins execution, by predeclaring it's read-set and write-set. If any of the predeclared data item can not be locked, the transaction does not lock any items; instead, it waits until all data items are available for locking.

2.6. Model Assumptions

We assume that the transactions are firm real time transactions. The model assumptions are listed below.

1) Processing of a transaction requires the use of CPU and data items located at local or remote site.

2) Arrival of the transactions at a site is independent of the arrivals at other sites and uses Poisson distribution.

3) Each transaction pre-declares its read-set (set of data items that the transaction will only read) and update-set (set of data items that the transaction will update).

4) The cohorts are executed in parallel.

5) A lending transaction cannot lend the same data item in read/update mode to more than one cohort to avoid cascaded abort.

6) The communication delay considered is either 0ms or 100ms to study the impact of the network delay on the system.

7) A distributed real time transaction is said to commit, if the coordinator has reached to commit decision before the expiry of the deadline at its site. This definition applies irrespective of whether cohorts have also received and recorded the commit decision by the deadlines [19].

8) Each cohort makes read and update accesses.

9) S2PL-HP is used for locking the data items.

10) Studies have been made for both main memory resident and disk resident database.

11) In case of disk resident database, buffer space is sufficiently large to allow the retention of data updates until commit time.

12) The updating of data items is made in transaction own memory rather than in place updating

3. Types of Dependencies

Sharing of data items in conflicting modes creates dependencies among conflicting transactions and constraints their commit order. We assume that a cohort requests an update lock if it wants to update a data item x. The prepared cohorts called as lenders lend uncommitted data to concurrently executing transactions known as borrower. Here, the borrower is further divided into two categories.

1) Commit Dependent Borrower

2) Abort Dependent Borrower

Therefore, modified definitions of dependencies used in this paper are given below:

Commit dependency (CD). If a transaction T_2 updates a data items read by another transaction T_1, a commit dependency is created from T_2 to T_1. Here, T_2 is called as commit dependent borrower and is not allowed to commit until T_1 commits.

Abort dependency (AD). If T_2 reads/updates an uncommitted data item updated by T_1, an abort dependency is created from T_2 to T_1. Here, T_2 is called as abort dependent borrower. T_2 aborts, if T_1 aborts and T_2 is not allowed to commit before T_1.

Each transaction/cohort T_i, that lends its data while in prepared state to an executing transaction/cohort T_j, maintains two sets.

1) Commit Dependency Set CDS (T_i): set of commit dependent borrower T_j, that are borrowed dirty data from lender T_i.

2) Abort Dependency Set ADS (T_i): the set of abort

dependent borrower T_j that are borrowed dirty data from lender T_i.

These dependencies are required to maintain the ACID properties of the transaction. Each lender is associated with a health factor defined as follows:

HF (health factor) = TimeLeft/MinTime

where TimeLeft is the time left until the transaction's deadline, and MinTime is the minimum time required for commit processing. The health factor is computed at the time when the coordinator is ready to send the YES-VOTE messages. MinHF is the threshold that allows the data held by committing transaction to be accessed. The variable MinHF is the key factor to influence the performance of the protocol. In our experiments, we have taken MinHF as 1.2, the value of MinHF used in PROMPT [12].

4. Condition for Fruitful Borrowing

If the deadline of an executing cohort is lesser than a committing cohort's commit time, then borrowing of locked data item is useless. Therefore, for fruitful borrowing, an incoming executing cohort having borrowing factor (BF) greater than a threshold value must be permitted to borrow the dirty data items from lender. The borrowing factor can be computed as given below.

Let us consider that transaction/cohort T_i that lends its data items while in prepared state to an executing transaction/cohort T_j, Here, T_i's voting phase is over and has entered in decision phase. The commit time (C_i) of T_i is the mean time required for the decision phase. It includes the time for sending the final commit message to the participating cohorts, the time for writing the final decision into stable storage, the time for permanently updating the data items for write operations and the time needed for releasing the locks [1]. *T*he minimum transaction response time (R_j) of T_j can be calculated as given below [17].

The deadline of a transaction is controlled by the runtime estimate of a transaction and the parameter slack factor, which is the mean of an exponential distribution of slack time. We allocate deadlines to arriving transacttions using the method given below. The deadlines of transactions (both global and local) are calculated based on their expected execution times [1,17].

The deadline (D_j) of transaction (T_j) is defined as:

$$D_j = A_j + SF * R_j$$

where, A_j is the arrival time of transaction (T_j) at a site; SF is the slack factor; R_j is the minimum transaction response time. As cohorts are executing in parallel, the R_j can be calculated as:

$$R_j = R_p + R_c$$

where, R_p, the total processing time during execution phase and commitment phase, and R_c, the communication delay during execution phase and commitment phase are given as below. For global transaction

R_p = max. $((2T_{\text{lock}} + T_{\text{process}})N_{\text{oper local}}, (2T_{\text{lock}} + T_{\text{process}})N_{\text{oper remote}})$

$R_c = N_{\text{comm}} T_{\text{com}}$

For local transaction

$R_p = (2T_{\text{lock}} + T_{\text{process}})N_{\text{oper local}}$

$R_c = 0$

Where, T_{lock} is the time required to lock/unlock a data item; T_{process} is the time to process a data item (assuming read operation takes same amount of time as write operation); N_{comm} is no. of messages; T_{com} is communication delay *i.e.* the constant time estimated for a message going from one site to another; $N_{\text{oper local}}$ is the number of local operations; $N_{\text{oper remote}}$ is maximum number of remote operations taken over by all cohorts. If T_2 is abort dependent on T_1.

The BF can be the ratio of $(SF*R_j - T_{\text{com}})/C_i$. For Fruitful Borrowing.

$$(SF*R_j - T_{\text{com}})/C_i > 1$$

5. ACTIVE Commit Protocol

5.1. Type of Dependency in Different Cases of Data Conflicts

Let T_1 be a transaction/cohort with identifier id_1 and holding a lock on data item *x*, and let T_2 be another transaction/cohort with id_2 requesting the same data item x, two situation may result depending on the status of T_1.

Situation 1: T_1 is a prepared cohort called as lender, lending uncommitted data. When data conflicts occur, there are three possible cases.

Case 1: Read-update conflict.

If T_2 requests an update-lock and it's BF > 1 while T_1 is holding a read-lock a commit dependency is defined from T_2 to T_1. First, the transaction id of T_2 is added to the CDS (T_1). Then T_2 acquires the update-lock

Case2: Update-Update conflict.

If T_2 requests an update-lock and it's BF > 1 while T_1 is holding an update-lock and HF(T_1) ≥ MinHF an abort dependency is defined from T_2 to T_1. The transaction id of T_2 is added to ADS (T_1), and T_2 acquires the update-lock; otherwise, T_2 is blocked.

Case 3: Update-Read conflict.

If T_2 requests a read-lock and it's BF > 1 while T_1 is holding an update-lock and HF(T_1) ≥ MinHF, an abort dependency is defined from T_2 to T_1. The transaction id of T_2 is added to ADS (T_1), and T_2 acquires the read-lock; otherwise, T_2 is blocked.

Situation 2: T_2 is commit dependent borrower going to become a lender simultaneously by lending its uncommitted data to an incoming transaction/cohort T_3. When

data conflicts occur, there are two possible cases of conflict.

Case 1: Update-Update conflict.

If T_3 requests an update-lock and it's BF > 1 while T_2 is holding an update-lock and HF(T_2) ≥ MinHF an abort dependency is defined from T_3 to T_2. The transaction id of T_3 is added to ADS (T_2), and T_3 acquires the update-lock; otherwise, T_3 is blocked.

Case 2: Update-Read conflict.

If T_3 requests a read-lock and it's BF > 1 while T_2 is holding an update-lock and HF(T_2) ≥ MinHF an abort dependency is defined from T_3 to T_2. The transaction id of T_3 is added to ADS (T_2), and T_3 acquires the read-lock; otherwise, T_3 is blocked.

In our Protocol, locks on data item are granted to those borrowers whose BF > 1. On the basis of the data conflicts discussed above, the access of data items in conflicting mode are processed by lock manager as follows.

If (T_1 is an independent lender)
{ If ((T_2 (BF > 1) CD T_1)
{ CDS (T_1) = CDS (T_1) ∪ {T_2};
T_2 is granted Update lock.
}
else if ((T_2(BF > 1) AD T_1) AND (HF(T_1) ≥ MinHF))
{ ADS (T_1) = ADS (T_1) ∪ {T_2};
T_2 is granted the requested lock (read or Update).
}
else T_2 will be blocked;
}
else if(T_1 is commit dependent borrower)
{ If ((T_2 (BF > 1) AD T_1) AND (HF (T_1) ≥ MinHF))
{ ADS (T_1) = ADS (T_1) ∪ {T_2};
T_2 is granted the requested lock (read or Update)
}
else T_2 will be blocked; }
else T_1 is not allowed to lend data

5.2. Mechanics of Interaction between Lender and Borrower Cohorts

Let us consider three transactions T_1, T_2 and T_3. T_2 has borrowed the data locked by T_1 being an independent lender. If T_2 is commit dependent borrower and has become a lender simultaneously by lending uncommitted data to an incoming transaction/cohort T_3, the following three scenarios are possible:

***Scenario 1*: T_1 receives decision before T_2 is going to start processing phase after getting all locks.**

T_1 is an independent lender

If the global decision is to commit, T_1 commits.

1) All cohorts in ADS (T_1) and CDS (T_1) will execute as usual and the sets ADS (T_1) and CDS (T_1) are deleted.

2) If the global decision is to abort, T_1 aborts. The cohorts in the dependency sets of T_1 will execute as follows:

All cohorts in ADS (T_1) will be aborted;

All cohorts in CDS (T_1) will execute as usual;

Sets ADS (T_1) and CDS (T_1) are deleted.

If, there is another commit dependent cohort T_2 on T_1 that has lent its dirty data to another cohort T_3, processing will be done as follows.

If T_1 aborts or commits, there are two possibilities with T_2.

1) It has either received the Yes-Vote message from its coordinator, or

2) It is still in wait state for the same.

In case of possibility 1, T_2's Yes-Response Message will be send to its coordinator. Type of Dependency between T_2 and T_3 will be governed by either Situation 1 Case 2 or Situation 1 Case 3 already discussed earlier depending on whether T_3 is update or read. In case of possibility 2, T_3 can not commit until T_2 terminates (*i.e.* commits or aborts) [21].

***Scenario 2*: T_2 is going to start processing phase after getting all locks before T_1 receives global decision.**

If T_1 is independent lender, T_2 is allowed to send a WORKSTARTED message to its coordinator, if it is commit dependent only; otherwise it is blocked from sending the WORKSTARTED message. It has to wait until

1) Alternative 1: either T_1 receives its global decisions, or.

2) Alternative 2: its own deadline expires, whichever, occurs earlier.

In Alternative 1, the system will execute as in the Scenario 1. In Alternative 2, T_2 will be killed and will be removed from the dependency set of T_1.

If, there is another cohort T_3 has borrowed dirty data from commit dependent borrower T_2, T_3 can not commit until T_2 terminates (*i.e.* commits or aborts).

***Scenario 3*: T_2 aborts before T_1 receives decision**

In this situation, T_2's and T_3's updates are undone and T_2 will be removed from the dependency set of T_1. T_2 and T_3 will be killed and removed from the system.

5.3. Algorithm

On the basis of above discussions, the complete pseudo code of the protocol is given below. if (T_1 (an independent lender) receives global decision before, T_2 is going to start processing phase after getting all locks)
{
ONE: if (T_1's global decision is to commit)
{
T_1 enters in the decision phase;

All cohorts in ADS (T_1) and CDS (T_1) will execute as usual;
Delete set of ADS (T_1) and CDS (T_1);
}
else //T_1's global decision is to abort
{
$T1$ aborts;
The cohorts in CDS (T_1) will execute as usual;
Transaction in ADS (T_1) will be aborted;
Delete sets of ADS (T_1) and CDS (T_1);
}
if (T_2 commit dependent on T_1 has lent its dirty data to another cohort T_3)
{
if (T_2 has received the Yes-Vote message from its coordinator) // T_1 has aborted or committed
{
T_2 send its Yes-Response Message its coordinator
T_2 & T_3 execute as two normal cohorts with T_3 dependent on T_1
}
else // T_2 is still waiting for Yes-Vote message
{
T_3 can not commit until T_2 terminates
}
}
}
else if (T_2 is going to start processing phase after getting all locks before, T_1(an independent lender) receives global decision)
{
check type of dependencies;
if (T_2's dependency is commit only)
{
T_2 sends WORKSTARTED message;
}
else
{
T_2 is blocked for sending WORKSTARTED message;
while (! (T_1 receive global decision OR T_2 misses deadline))
{
if (T_2 misses deadline)
{
Undo computation of T_2;
Abort T_2;
Delete T_2 from CDS (T_1) & ADS (T_1);
}
else GoTo ONE;
}
}
if (T_3 has borrowed dirty data item from commit dependent borrower T_2)
T_3 cannot commit until T_2 terminates
}
else //T_2 is aborted by higher transaction before,
// T_1 receives decision
{
Undo computation of T_2 & T_3;
Abort T_2 & T_3;
Delete T_2 from CDS (T_1) & ADS (T_1);
}

5.4. Main Contributions

1) Borrower cohorts are categorized as commit or abort dependent.

2) Commit dependent borrowers are allowed to further lend data, thereby reducing the data inaccessibility. The length of cascaded abort chain is same as in the case of PROMPT.

3) Borrowing factor is computed for each borrower, so that lock on requested dirty data items can be granted to borrowers in fruitful way only.

To maintain consistency of database, cohort sends the YES-VOTE in response to its coordinator's VOTE-REQ message only when its dependencies are removed and it has finished its processing.

6. Performance Measures and Evaluation

The default values of different parameters for simulation experiments are same as in [17]. The concurrency control scheme used is static two phase locking with higher priority. Miss Percentage (MP) is the primary performance measure used in the experiments and is defined as the percentage of input transactions that the system is unable to complete on or before their deadlines [19].

Since there were no practical benchmark programs available in the market or with research communities to evaluate the performance of protocols and algorithms, an event driven based simulator was written in C language. In our simulation, a small database (200 data items) is used to create high data contention environment. For each set of experiment, the final results are calculated as an average of 10 independents runs. In each run, 100000 transactions are initiated.

6.1. Simulation Results

Simulation was done for both the main memory resident and the disk resident databases at communication delay of 0ms and 100ms. We compared ACTIVE with PROMPT, 2SC and SWIFT in this experiment. **Figure 2** to **Figure 6** show the Miss Percent behavior under normal and heavy load conditions with/without communication delay. **Figure 2** and **Figure 3** deal with main memory based database system while rest of the figures deal with disk resident database system. In these graphs, we

first observe that there is a noticeable difference between the performances of the various commit protocols throughout loading range. This is due to careful lending of data to a borrower. Here, commit dependent borrowers has also lent data to executing cohort reducing the data inaccessibility. At the same time, the transaction abort chain restricts to one only. Also, an incoming executing cohort having borrowing factor greater than one has been permitted only to borrow the dirty data items from lender. Thus, the work done by the borrowing cohort is never wasted because of better borrowing choice. This minimizes the fruitless borrowing by the cohort. In general the number of transaction being committed are more than number of aborted transactions in real life situations. In this way, we can increase some more parallelism in the distributed system. The ACTIVE commit protocol provides a performance that is significantly better than other commit protocols.

Main Memory Resident Database

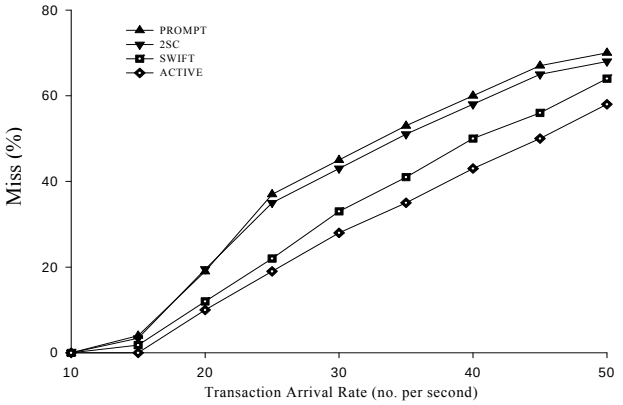

Figure 2. Miss % with (RC + DC) at communication delay 0ms normal and heavy load.

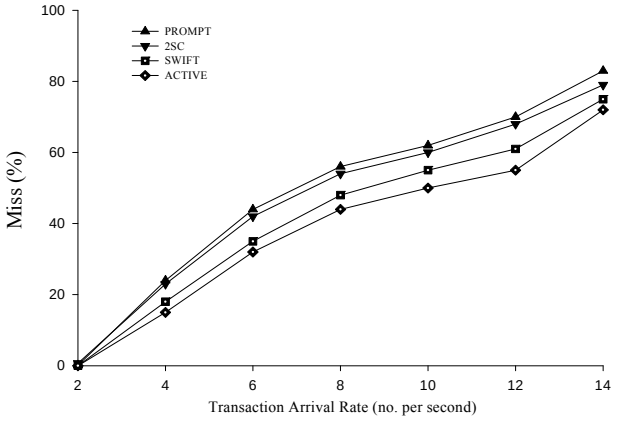

Figure 3. Miss % with (RC + DC) at communication delay 100ms normal and heavy load.

Disk Resident Database System

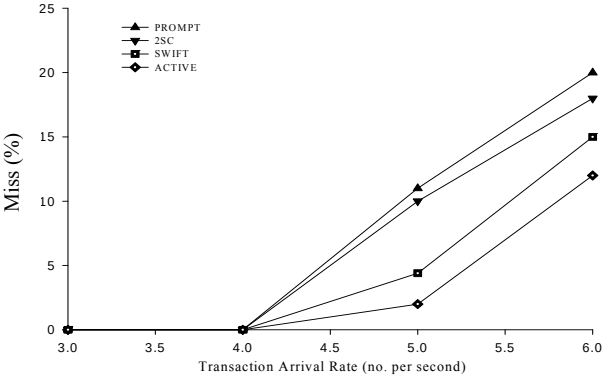

Figure 4. Miss % with (RC + DC) at communication delay 0ms normal load.

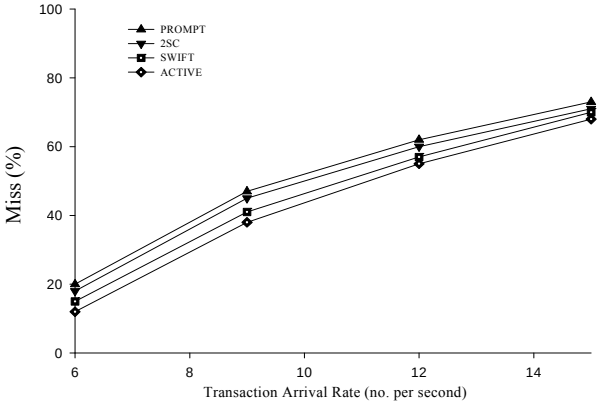

Figure 5. Miss % with (RC + DC) at communication delay 0ms heavy load.

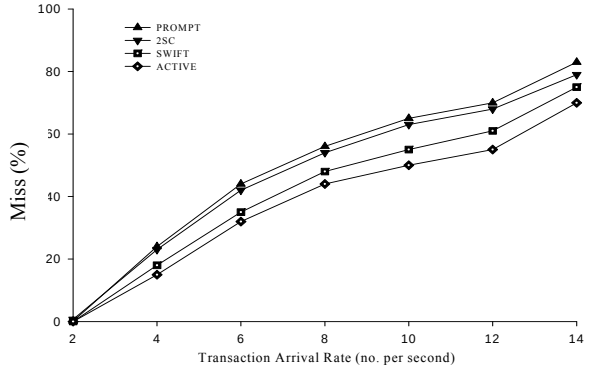

Figure 6. Miss % with (RC + DC) at communication delay 100ms normal and heavy load.

7. Future Research Directions

Following are some suggestions to extend this work [13,20].

1) Despite much research in the area of wireless sensor networks in recent years, the programming of sensor

nodes is still time-consuming and tedious. A new paradigm which seems to be qualified to simplify the programming of sensor networks is the Service Oriented Architecture. The composition of simple services to more complex ones can be a convenient way to design applications. To enable the sophisticated techniques known from service oriented architectures like replication and migration of services, a transaction model for sensor networks is required. The paper [22] studied the applicability of the standard commit protocols Two Phase Commit (2PC) and Transaction Commit on Timeout and show in experiments with real sensor nodes that 2PC can enable consistent service migration in wireless sensor networks. This study can further be extended with this protocol.

2) Our performance studies are based on the assumption that there is no replication. Hence, a study of relative performance of the topic discussed here deserves a further look under assumption of replicated sensor databases.

3) The work can be extended for Mobile DRTDBS, pear-to-pear database systems, grid database systems etc. where memory space, power and communication bandwidth are bottleneck. There is a need to design various protocols for different purposes that may suit to the specific need of hand held devices.

4) Biomedical Informatics is quickly evolving into a research field that encompasses the use of all kinds of biomedical information, from genetic and proteomic data to image data associated with particular patients in clinical settings. Biomedical Informatics comprises the fields of Bioinformatics (e.g., genomics and proteomics) and Medical Informatics (e.g., medical image analysis), and deals with issues related to the access to information in medicine, the analysis of genomics data, security, interoperability and integration of data-intensive biomedical applications. Main issues in this field is provision of large computing power such that researchers have access to high performance distributed computational resources for computationally demanding data analysis, e.g., medical image processing and simulation of medical treatment or surgery and large storage capacity and distributed databases for efficient retrieval, annotation and archiving of biomedical data. What is missing today is full integration of methods and technologies to enhance all phases of biomedical informatics and health care, including research, diagnosis, prognosis, etc. and dissemination of such methods in the clinical practice, whenever they are developed, deployed and maintained. Hence it is another topic of research interest.

8. Conclusions

This paper proposes a new commit protocol ACTIVE for DRTDBS, where the commit dependent borrower is allowed to lend its data to an incoming cohort. It reduces the data inaccessibility by allowing the commit dependent borrowers to further act as lenders. Also, to increase the benefit of borrowing, an incoming cohort having borrowing factor greater than one can borrow the dirty data items from a lender. This reduces the fruitless borrowing. To ensure non-violation of ACID properties, checking of the removal of cohort's dependency is required before sending the Yes-Vote message. The performance of ACTIVE is compared with other protocols for both main memory resident and disk resident databases with and without communication delay. Simulation results show that the proposed protocol improves the system performance.

As future work, it is desirable to study the performance of the proposed protocol in real environment.

9. References

[1] Y. Lam, C-L. Pang, S. H. Son, and J. Cao, "Resolving executing-committing conflicts in distributed real-time database systems," Computer Journal, Vol. 42, No. 8, pp. 674–692, 1999.

[2] C.-L. Pang and K. Y. Lam, "On using similarity for resolving conflicts at commit in mixed distributed real-time databases," Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications, 1998.

[3] G. K. Attaluri and K. Salem, "The presumed-either two-phase commit protocol," IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No. 5, pp. 1190–1196, 2002.

[4] J. Gray and A. Reuter, "Transaction processing: Concepts and technique," Morgan Kaufman, San Mateo, California, 1993.

[5] J. Gray, "Notes on database operating systems," Operating Systems: An Advanced Course, Lecture Notes in Computer Science, Springer Verlag, Vol. 60, pp. 397–405, 1978.

[6] P. Misikangas, "2PL and its variants," Seminar on Real-Time Systems, Department of Computer Science, University of Helsinki, 1997.

[7] I. Lee and Y. H. Yeom, "A single phase distributed commit protocol for main memory database systems," 16th International Parallel & Distributed Processing Symposium (IPDPS), Ft. Lauderdale, Florida, USA, 2002.

[8] C. Mohan, B. Lindsay, and R. Obermarck, "Transaction management in the R* distributed database management system," ACM transaction on Database Systems, Vol. 11, No. 4, 1986.

[9] N. Soparkar, E. Levy, H. F. Korth, and A. Silberschatz, "Adaptive commitment for real-time distributed transaction," Technical Report TR-92–15, Department of Computer Science, University of Texax, Austinm, 1992.

[10] R. Gupta, J. R. Haritsa, and K. Ramamritham, "More optimism about real-time distributed commit processing," Technical Report TR–97-Database System Lab, Super-

computer Education and Research Centre, Indian Institute of Science, Bangalore, India, 1997.

[11] R. Gupta, J. R. Haritsa, K. Ramamritham, and S. Seshadri, "Commit processing in distributed real time database systems," Proceedings of Real-Time Systems Symposium, IEEE Computer Society Press, Washington DC, San Francisco, 1996.

[12] J. R. Haritsa, K. Ramamritham, and R. Gupta, "The PROMPT real time commit protocol," IEEE Transaction on Parallel and Distributed Systems, Vol. 11, No. 2, pp. 160–181, 2000.

[13] U. Shanker, M. Misra, and A. K. Sarje, "Distributed real time database systems: Background and literature review," International Journal of Distributed and Parallel Databases, Springer Verlag, Vol. 23, No. 2, pp. 127–149, 2008.

[14] B. Qin and Y. Liu, "High performance distributed real time commit protocol," Journal of Systems and Software, Elsevier Science Inc., pp. 1–8, 2003.

[15] K. Ramamritham and P. K. Chrysanthis, "A taxonomy of correctness criteria in data-base applications," Journal of the Very Large Data Bases, Vol. 5, pp. 85–97, 1996.

[16] B. Qin, Y. Liu, and J. Yang, "A commit strategy for distributed real-time transaction," Journal of Computer Science and Technology, Vol. 18, No. 5, pp. 626–631, 2003.

[17] U. Shanker, M. Misra and A. K. Sarje, "SWIFT - a new real time commit protocol," International Journal of Distributed and Parallel Databases, Springer Verlag, Vol. 20, No. 1, pp. 29–56, 2006.

[18] U. Shanker, M. Misra, A. K. Sarje and R. Shisondia, "Dependency sensitive shadow SWIFT," Proceedings of the 10th International Database Applications and Engineering Symposium, Delhi, India, pp. 373–276, 2006.

[19] O. Ulusoy, "Concurrency control in real time database systems," PhD Thesis, Department of Computer Science, University of Illinois Urbana-Champaign, 1992.

[20] U. Shanker, M. Misra, and A. K. Sarje, "Hard real-time distributed database systems: Future directions," Communication Network, Department of Electronics & Computer Engineering, Indian Institute of Technology Roorkee, India, pp. 172–177, 2001.

[21] D. Agrawal, A. El Abbadi, R. Jeffers and L. Lin, "Ordered shared locks for real-time databases," International Journals of Very Large Data Bases, Vol. 4, No. 1, pp. 87-126, January 1995.

[22] Christoph Reinke, Nils Hoeller, Jana Neumann, Sven Groppe, Volker Linnemann and Martin Lipphardt, "Integrating standardized transaction protocols in service-oriented wireless sensor networks," Proceedings of the 2009 ACM symposium on Applied Computing, Honolulu, Hawaii, pp. 2202-2203, 2009

