FIVE: A Real-Time Commit Protocol

Rabin Kumar Singh
Department of Computer & Engineering
M.M.M.Engg.College
Gorakhpur,India

Pradeep Kr. Baitha
Department of Computer & Engineering
M.M.M.Engg.College
Gorakhpur,India

Vishal Pathak
Department of Computer & Engineering
M.M.M.Engg.College
Gorakhpur,India

Dr. Udai Shanker
Department of Computer & Engineering
M.M.M.Engg.College
Gorakhpur,India

ABSTRACT
In recent years, numerous commit protocols have been proposed for the lending of prepared data to the borrower in its commit phase to overcome the problem of data inaccessibility, but few of these considered the fruitful borrowing of this data. In this paper we have proposed a new commit protocol for distributed real time database systems (DRTDBS) by investigating lender-borrower relation in detail and also considered such systems which have combination of many non-healthy borrowers and super-healthy borrowers. Fruitful lending of Incredible Value added data without Extending abort chain (FIVE), which considered all types of borrowers and lend the data in a fruitful way and solve the problem of higher kill percentage of transactions and we categorized the borrower cohorts 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. This minimizes the fruitless borrowing by the cohort. The performance of FIVE is compared with ACTIVE, PROMPT, 2SC and SWIFT protocols for both main memory resident and disk resident databases without communication delay for the soft real time distributed transactions.

Keywords
Distributed Real Time Database Systems (DRTDBS), Commit Protocol, Conflict Resolution, Dependency, Lender, Borrower, Modified-Borrow Factor (M-BF)

1. INTRODUCTION
Database systems are currently being used as backbone to thousands of applications, which have very high demands for availability and fast real-time responses. Today's Real-time database systems(RTDBS) operating on distributed data have to contend with the well-known complexities of supporting transaction ACID semantics in the distributed environment and also data conflicts are one of the most important factors amongst the transactions. Two kinds of conflicts between transactions arise. One occurs between executing transactions, and can be resolved by a concurrency control protocol to ensure distributed transaction serializability; the other occurs between executing-committing transactions, which can be resolved by a commit protocol to ensure distributed transaction atomicity. Till now limited work have been noticed in executing-committing conflicts cases.

Database system plays a measure role in the present scenario, 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 in time 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. 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. So to solve this problem we need to develop a better commit protocol for DRTDBS.

Numerous protocols have been proposed to solve these problems of data conflictions. 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. 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 commit protocols have been proposed and can be classified into main following four groups:

- *Presumed Abort/Presumed Commit*
- *One Phase*
- *Group Commit*
- *Pre Commit/Optimistic*

Soparkar et *al*. have proposed a protocol that allows individual site to unilaterally commit. Gupta et *al*. proposed optimistic commit protocol and its variants. Presumed commit (PC) and presumed abort (PA) are based on 2PC. Enhancement has been made in PROMPT commit protocol, 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 cannot 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. 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.

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

(b) reversing the commit dependency

When a transaction, which depends on a committing transaction, wants to enter in the decision phase and its deadline is approaching.

To overcome the problem of DDCR, Pang C.-L. and Lam K. Y. 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. Y.Liu et *al*. proposed double space commit (2SC) protocol which based On PROMPT and DDCR. They analyzed and categorized all kind of dependencies that may occur due to data access conflicts between the transactions into two types commit dependency end abort dependency. The 2SC protocol allows a non-healthy transaction to lend its held data to the transactions in its commit dependency set.

Udai Shanker et *al*. proposed SWIFT protocol. In SWIFT, the execution phase of a cohort is divided into two parts, locking phase and processing phase and 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 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.

U. Shanker et *al.* proposed ACTIVE to study the both disk resident as well as data in memory and also lend the data based on Borrow factor to solve the problem of data inaccessibility.

In this paper we have proposed a new protocol FIVE-A Real Time Commit Protocol, which allows lender to lend data according the scenario at the time of lending data, which reduced the kill percentage of transactions and also beneficial both for main memory and disk resident databases.

2. BACKGROUND AND RELATED WORK

As deadline of transaction a play a significant role in the real-time transactions. For the study of these type of transaction, distributed real-time database system (DRTDBS) model used.

2.1 DRTDBS Model:

This model includes the description of its various components such as network model, system model, database model, cohort execution model; locking mechanism .The common model for DRTDBS is given below in **Figure**-1. At each site, two types of transactions are generated:

- global transactions
- local transactions

Each global transaction consists of m cohorts, where m is less than or equal to the number of database sites Nsite. 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 Noper number of database operations. Each operation requires locking of data items and then processing.

2.1.1 Network Model

All sites communicate via messages exchange over the communication network because it has no global shared memory in the system. Thus, a network manager models the behavior of the communications network.

2.1.2 Cohort Execution Model

Distributed execution model can be categorized into two types, as:

- sequential
- parallel

Sequential execution model consist at most one cohort of a transaction at each execution site, and only one cohort can be activated at a time. After the successful completion of one operation, the next operation in the sequence is executed by the appropriate cohort and at the end of the execution of the last operation, the transaction can be committed. But in case of

parallel execution model, the coordinator of the transaction spawns cohorts all together by sending a message to remote sites with a request to activate the cohort, lists all operations to be executed at that site and then all 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 the other cohorts at some other sites. It shows that, the sibling cohorts do not have to require any information from each other to share. Here we have considered cohorts executing in parallel way.

2.1.3 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.

2.1.4 Database Model

This model is the collection of data items that are uniformly distributed over 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.

2.1.5 Locking Mechanism

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. Locks are means for synchronizing the access of concurrent transactions to the database items. When all locking operations precede the first unlock operation in the transaction, this locking technique said to be two-phase locking. 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 pre-declaring it's read-set and write-set. If any of the pre-declared data item cannot be locked, the transaction does not lock any items; instead, it waits until all data items are available for locking, after the execution will start.

2.1.6 Model assumptions

For the study of real-time transactions, in this paper following assumptions used:

- 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.
- Arrival of the transactions at one site is independent of the arrivals at other sites and uses Poisson distribution.
- For locking the data items S2PL-HP is used.
- Processing of a transaction requires the use of CPU and data items located at local or remote site.
- The updating of data items is made in transaction own memory rather than in place updating.
- A lending transaction cannot lend the same data item in read/update mode to more than one cohort to avoid cascaded abort.
- The communication delay considered is either 0ms or 100ms to study the impact of the network delay on the system.
- Each cohort makes read and update accesses.
- Studies have been made only for both main memory and disk resident database.
- The cohorts are executed in parallel mode.
- 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).

3. 2SC COMMIT PROTOCOL

2SC allows two transactions to share the data by allowing a transaction (borrower) to borrow the data from a transaction in its commit phase (lender). Two types of dependencies created when data items shared in conflicting mode:

- Commit dependency
- Abort dependency

Commit dependency (CD: If a transaction T2 updates a data items read by another transaction T1, a commit dependency is created from T2 to T1. Here, T2 is called as commit dependent borrower and is not allowed to commit until T1 commits.

Abort dependency (AD): If T2 reads/updates data item that is not updated by T1, an abort dependency is created from T2 to T1 and T2 is called as abort dependent borrower. T2 aborts, if T1 aborts and T2 is not allowed to commit before T1.

Here each transaction/cohort Ti, that lends its data while in prepared state to an executing transaction/cohort Tj creates two sets:

- Abort Dependency Set ADS
- Commit Dependency Set CDS

Commit Dependency Set CDS (Ti): set of commit dependent borrower Tj that are borrowed dirty data from lender Ti.

Abort Dependency Set ADS (Ti): the set of abort dependent borrower Tj that are borrowed dirty data from lender Ti. These dependencies are required to maintain the ACID properties of the transaction.

When T2 had accessed the locked data, three situations may arise:

- *Read–WRITE Conflict:* If T2 requests a write-lock while T1 is holding a read-lock a commit dependency is defined from T2 to T1. First, the transaction id of T2 is added to the CDS (T1). Then T2 acquires write lock.
- *Write-Write Conflict:* If both locks are update locks, a commit dependency is defined from T2 to T1. After the transaction id of T2, is added to the CDS (T1), T2 acquires the write-lock.
- *Write –Read Conflict*. If T2 requests a read lock while T1 is holding a update-lock, an abort dependency is defined from T2 to T1. If HF(T1) > Min-HF, the transaction id of T2, is added to the ADS (T1) then T2 acquires the write-lock; Otherwise, If HF(T1) < Min-HF, T2 is blocked.

Also when T2 had accessed the locked data, three situations may arise:

Situation 1: T1 receives decision before T2 has completed its local processing-

If the global decision is to commit, T1 commits. All transactions in ADS (T1) and CDS (T1) will execute as usual and the set of ADS (T1) and CDS (T1) will be deleted.

When decision is global to commit, T1 commits.

- All the cohorts in ADS (T1) and CDS (T1) will execute as usual and the sets ADS (T1) and CDS (T1) are deleted.
- In case the global decision is to abort, T1 will aborts. The cohorts in the dependency sets of T1 will execute as follows:

a. All cohorts in ADS (T1) will be aborted.

b. All cohorts in CDS (T1) will execute as usual.

c. Sets ADS (T1) and CDS (T1) are deleted.

Situation 2: T2 is going to start its processing phase before T1 receives global decision:

Here T2 is not allowed to send a WORKDONE message to its lender and allowed to send a WORKSTARTED message to its coordinator, if it is commit dependent only. It has to wait until:

- Either T1 receives its global decisions, or abort.
- Its own deadline expires, whichever occurs earlier.

In first case, the system will execute as in the Scenario 1. In second case, T2 will be killed and will be removed from the dependency set of T1. If, there is another cohort T3 has borrowed dirty data from commit dependent borrower T2, T3 can not commit until T2 terminates (i.e. Commits or aborts).

Situation 3: T2 aborts before T1 receives decision:

In this situation, T2's updates are undone and T2 will be removed from the dependency set of T1.

Each lender is associated with a health factor defined as follows:

HF (health-factor) = Time-Left/Min-Time

Where Time-Left is the time left to meet the transaction's deadline, and Min-Time is the minimum time required to complete the commit processing. The health factor is computed at the time when the coordinator is ready to send the YESVOTE messages. Min-HF is the threshold that allows the data held by committing transaction to be accessed. The variable Min-HF is the key factor to influence the performance of the protocol. In our experiments, we have taken Min-HF as 1.2, the value of Min-HF used in PROMPT.

4. BORROWING CONCEPT AND BORRROW FACTOR

To solve the problem of data inaccessibility some protocols allows the committing transactions to lend data to the transactions requesting data items held by that transactions. That is prepared cohorts lend their uncommitted data to concurrently executing transactions. This creates interaction between Lender and Borrower and to avoid cascading abort because this lending chain is limited to one. Also to find the borrower concept of borrower factor is used. 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. Consider that transaction/cohort Ti that lends its data items while in prepared state to an executing transaction/cohort Tj, Here, Ti's voting phase is over and has entered in decision phase. The commit time (Ci) of Ti 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.

To control the deadline of a transaction we need estimated runtime of a transaction and the parameter slack factor, which is the mean of an exponential distribution of slack time. We allocate deadlines to arriving transactions using the method given below. The deadlines of global and local are calculated based on their expected execution times. The deadline (Dj) of transaction (Tj) is defined as:

$$Dj = Aj + SF * Rj$$

Where Aj is the arrival time of transaction (Tj) at a site; SF is the slack factor and Rj is the minimum transaction response time. As cohorts are executing in parallel, the Rj can be calculated as**:**

$$Rj = Rp + Rc$$

Where Rp used for total processing time during execution phase and commitment phase, and Rc for the communication delay during execution phase and commitment phase and computed as:

For global transactions-

Rp = max((2Tlock + Tprocess)Noper local,(2Tlock + Tprocess)Noper remote)

Rc = NcommTcom

For local transactions-

Rp = (2Tlock + Tprocess)Noper local

Rc = 0

Where, Tlock is the time required to lock/unlock a data item; Tprocess is the time to process a data item (assuming read operation takes same amount of time as write operation); Ncomm is number of messages; Tcom is communication delay i.e. the constant time estimated for a message going from one site to another; Noper local is the number of local operations; Noper remote is maximum number of remote operations taken over by all cohorts. If T2 is abort dependent on T1. The BF can be the ratio of (SF*Rj-Tcom)/Ci;

BF=(SF*Rj-Tcom)/Ci>1

U.Shanker et *al*. used 1(one) as the thresholds value for BF in ACTIVE-a real-time commit protocol. In this paper we have also taken the same value for the study.

5. FIVE COMMIT PROTOCOL

In this paper we have proposed a new commit protocol to reduce the kill percentage of transactions by fruitful borrowing of prepared data of lender to a executing cohort in its commit phase. Here we have considered the case of parallel transactions in distributed real time database system. In this paper real time transactions have divided (assumptions) into three sections based on borrowing factor as shown in **Figure-**2:

- *Non-healthy transactions*: these transactions have borrowing factor (BF) less than one(i.e threshold value of BF).
- *Healthy transactions:* these transactions have BF between one and less than the Min-Time (MT).
- *Super-healthy transactions:* these transactions have BF greater than Min-Time

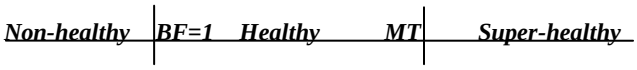

Figure-2

To lend the data as per the condition at the time of lending, FIVE considering the following cases:

Case 1: When Non-healthy transactions more than the both Healthy and Super-healthy transactions:

In this case data borrowing carried out by using the modified-Borrow factor (M-BF) as-

M-BF←BF/Time-Left*

Where Time-Left* is that time lift to meet the deadline for borrower. Higher the value of M-BF higher the priority of transaction Ti that is P[Ti] , higher the to borrow the prepared data as:

P[Ti]←M-BF[Ti]

Higher the value of M-BF, higher the priority to borrow the dirty value from the lender in its commit phase.

Case 2: When Healthy transactions more than the both Non-healthy transactions and Super-healthy transactions:

Borrowing carried out by using the borrow factor as -

P[Ti]←BF[Ti]

Where P[Ti] is the priority of transaction Ti and BF[[Ti] is the borrow factor of transaction Ti, higher the value of BF, higher the priority to borrow the dirty value.

Case 3: When Super-healthy transactions more than the both Non-healthy and Healthy transactions:

In this case lender has been updated its data to the stable storage. Due to this their possibility of aborting the executing cohorts, this will affect the system performance by increasing the kill percentage of transactions. To solve this problem, need to fork a replica of prepared data and carried out the execution of transaction by using the replica of prepared data as well as by using prepared data. In case of lender abort then execution will carried out by using the replica and borrow the data by using BF. Where borrow factor is same as calculated in section 4 above. Time-left give the execution time of borrower. Priority of these transactions computed similar to case 2, but lending is carried out by using replica of dirty value:

P[Ti]←BF[Ti]

Where P[Ti] is the priority of transaction Ti and BF[[Ti] is the borrow factor of transaction Ti. Higher the BF higher the priority to borrow the dirty value.

Heuristics for lending the prepared data:

Consider two borrowers T1 and T2, we calculate Modified-Borrow Factor for both and compare them.

(a) *If(BF<1)*

```
{
    If  (M-BF(T1) > (M-BF(T2))
  {
       then T1 is allowed to borrow
  }
     Else

  { T2 allowed to borrow the prepared data.
  }
}
```

(b) *If((BF>1)AND(BF>Min-Time))*

```
{
   { If ( (BF (T1) >BF(T2))
{
       then T1 is allowed to borrow
}
 else
{
T2 allowed to borrow the prepared data.
}
```

(c) *If (BF>Min-Time)*

```
{
Lend the data by using replica of prepared data.
    If  ((BF (T2)< (BF (T1))
        {
        then T1 is allowed to      borrow .
        }
       else
      {
        T2 allowed to lend the data.
      }
 }
```

5. MAIN CONTRIBUTIONS

- FIVE overcomes the problem of data inaccessibly with reducing the kill percentage of transactions.
- Increase the system performance by effective borrowing the prepared data.
- Modified health factor used to find the BF when the BF is less than the one.
- This paper also solves the problem of Super-healthy transactions by lending the data by using replica of prepared data.
- These protocols also control the abort chain of executing transactions.

6. RESULT AND ANALYSIS

In this paper we have done the experiment both for disk resident as well as for main-memory resident data by considering the communication delay 0ms. FIVE is compared with ACTIVE, PROMPT, 2SC and SWIFT**. Figure**-3 **and Figure**-4 shows the Miss percent behavior under normal and heavy load. **Figure-3** deals with Main-Memory based database system and **Figure-4** deals with the disk resident database system. In these graphs we have observed that there are some differences between the performances of various commit protocols at the time lending of data.

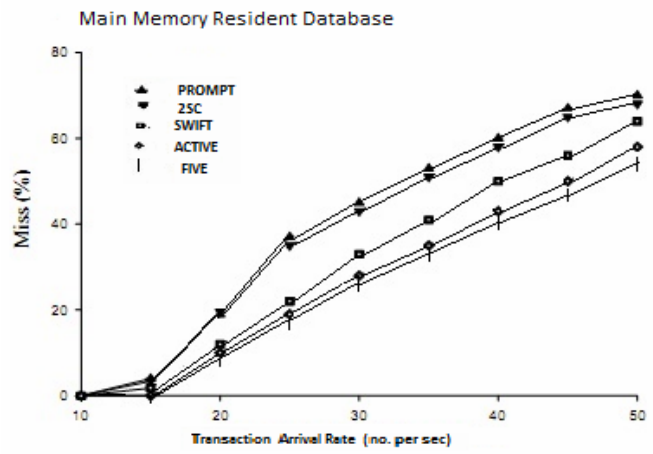

Figure 3: Main % with(RC+DC) at communication delay 0ms normal and heavy load

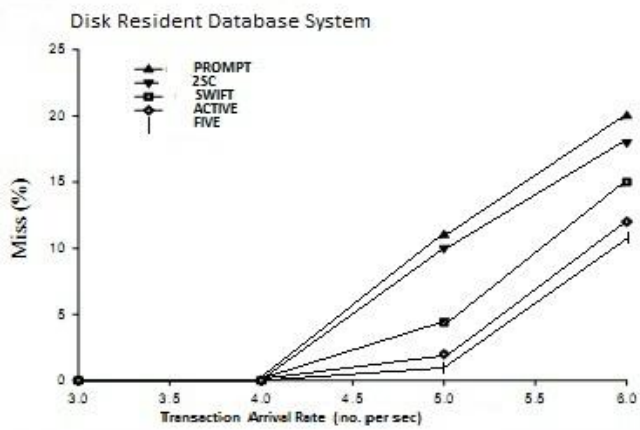

Figure 4: Miss % with (RC+DC) at communication delay 0ms normal load

This is due to careful lending of data to a borrower. We have assumed that commit dependent borrowers also lent data to executing cohort reducing the data inaccessibility also the transaction abort chain restricts to one only. Incoming executing cohorts allow to borrow the data according to the situation at the time of borrowing of prepared data. Thus, the work done by the borrowing cohort is never wasted because of better borrowing choice. Hence it will minimize the fruitless borrowing by the cohort. Therefore we can say that the number of transaction being committed is more than number of aborted transactions in real life situations. In this way, we can increase some more parallelism in the distributed system. The FIVE commit protocol provides a performance that is significantly better than other commit protocols.

7. CONCLUSION

In this paper we have proposed a new commit protocol FIVE. This protocol basically solves the problem of kill transactions by calculating the BF at the time of lending data items. 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. This protocol also overcomes the problem of ACTIVE which considered only that borrower which have borrow factor greater than a threshold value (i.e. 1). In this paper transactions categorized into three sections according to the borrowing factor and lend the data to those transactions which are more than from other two cases. By this way system performance can be increases. The performance of FIVE is compared with other protocols for both main memory resident and disk resident databases without communication delay.

REFERENCES

[1] U. Shanker et al."ACTIVE-a real time commit protocol" Wireless Sensor Network, 2010, 2, 254-263..

[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] 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.

[8] R. Gupta, J. R. Haritsa, and K. Ramamritham, "More optimism about real-time distributed commit processing,".Technical Report TR–97-Database System Lab, SuperU. Shanker *ET AL*. Copyright © 2010 SciRes. ***WSN***

[9] 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.

[10] 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.

[11] 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.

[12] 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 .

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

[14] 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.