

# CCRTRD :A Protocol for Concurrency Control in Real-Time Replicated Databases System

Ashish Srivastava<sup>1</sup>

Deptt. Of Computer Application

M.B.S.P.G. College, Gangapur, Varanasi, India  
[ashish24.vns@gmail.com](mailto:ashish24.vns@gmail.com)<sup>1</sup>

Udai Shankar<sup>2</sup>

Deptt. Of Computer Sc. & Engineering

M.M.M. University of Technology, Gorakhpur, India  
[udaigkp@gmail.com](mailto:udaigkp@gmail.com)<sup>2</sup>

**Abstract**— Replication is the technique of using multiple copies of a server can help database systems meet the stringent temporal constraints of current time-critical applications, The main goal of replication [5] is to improve availability and consistency since a service is available even if some of its replicas are not. This helps mission critical services, such as many financial systems or reservation systems, In fact, if the system is not designed to handle overloads, the effects can be catastrophic and some primordial transactions of the application can miss their deadlines.

We propose a new Concurrency Control protocol in replicated distributed environment which is especially for High Priority point and firm real time database system. A Concurrency control in real time replicated databases (CCRTRD) mechanism uses S2PL (Static Two Phase Locking) for deadlock free environment. It also includes High Priority given to a cohort after receiving PREPARE message from its coordinator. Also with some more assumptions like sending an extra message in execution phase but after completion of execution at local copy which is described later in this paper the proposed mechanism has a significant increased performance over O2PL and MIRROR in decreasing execution time of the current transaction and it also decreases the waiting time of transactions in pass the time queue.

**Keywords-** *Replicated database, Distributed Databases systems Real Time system, two-phase commit, Concurrency control*

## I. INTRODUCTION

The Existing applications, such as Web-based services, electronic commerce, mobile telecommunication system, etc., are distributed in nature and manipulate time-critical databases. In order to enhance the performance and the availability of such applications, one of the main techniques is to replicate data on multiple sites of the network. Therefore, the major issue is to develop efficient replica concurrency control protocols that are able to tolerate the overload of the distributed system. In fact, if the system is not designed to handle overloads, the effects can be catastrophic and some primordial transactions of the application can miss their deadlines. While many efforts have been made in the management of transactions for replicated databases in the real-time context [5,6,7], no work deals with protocols that manage distributed real-time databases and simultaneously control the overload of the system. Some researches have dealt with the scheduling of tasks under overload conditions when the real-time system is

centralized such as in [5] distributed as in [2]. Distributed real time database systems (DRTDBSs) can be defined as database systems that support real time transactions. A distributed database is a single logical database that is spread physically across computers in multiple locations that are connected by a data communications network. The network must allow the users to share the data i.e. user at location A must be able to access the data at location B. They are used for a wide spectrum of applications such as air traffic control, stock market trading, banking, telemedicine etc. In DRTDBS, prior to, they have resulted in schemes wherein either the standard notions of database correctness are not fully supported, or the maintenance of multiple historical versions of the data is required, or the real-time transaction semantics and performance metrics pose practical problems. Further, none of these studies have considered the optimistic two-phase locking (O2PL) protocol [4] although it is the best-performing algorithm in conventional (non-real-time) replicated database systems [4]. Transactions in a real time database are classified into three types, viz. hard, soft and firm. The classification is based on how the application is affected by the violation of transaction time constraints. This paper reports efficient solutions for some of the issues important to the performance of replicated firm deadline based DRTDBS [9,10]. The performance of DRTDBS depends on several factors such as specification of transaction's deadline, priority assignment policy, scheduling transactions with deadlines, time conscious buffer and locks management, commit procedure etc. One of the primary performance determinants is the policy used to schedule transactions for the system resources. The resources that are typically scheduled are processors, main memory, disks and the data items stored in database. On possible goal of replication is to have replicas behave functionally like no replicated servers. This goal can be stated precisely by the concept of one-copy serializability, which extends the concept of serializability to a system where multiple replicas are present. An execution is one-copy serializable if it has the same effect as a serial execution on a one-copy database. We would like a system to ensure that its executions are one-copy serializable. In such a system, the user is unaware that data is replicated. There are two approaches to sending a transaction's updates to replicas: synchronous and asynchronous. In the synchronous approach, when a transaction updates a data item, say x, the update is sent to all replicas of x. These updates of the replicas execute within the context of the transaction. This

is called synchronous because all replicas are, in effect, updated at the same time. Although sometimes this is feasible, often it is not, because it produces a heavy distributed transaction load. In particular, it implies that all transactions that update replicated data have to use two-phase commit, which entails significant communications cost. Fortunately, looser synchronization can be used, which allows replicas to be updated independently. This is called asynchronous replication, where a transaction directly updates one replica and the update is propagated to other replicas later. In a replicated environment means when there is more than one copy of a data item distributed on different sites then there is one major issue how to update replicas of data item on different site with efficient concurrency control mechanism. While few works have been done in area of replica concurrency control like 2PL, O2PL and OCC but they are not up to the standard of present requirement.

In this paper we focus on concurrency control and one copy serializability in replicated firm real time database system. In firm real time system the major issue is deadline of completion means if the transaction misses its deadline then the transaction is of no usage, hence it is killed if the transaction misses its deadline.

## II. ASSOCIATED EFFORT

Concurrency control algorithms and real-time conflict resolution mechanisms for RTDBS have been studied extensively (e.g. [10, 11]). However, concurrency control for replicated DRTDBS. An algorithm for maintaining consistency and improving the performance of replicated DRTDBS is proposed in [20]. In this algorithm, a multiversion technique is used to increase the degree of concurrency. Replication control algorithms that integrate real-time scheduling and replication control are proposed in [6, 7]. These algorithms employ Epsilon-serializability (ESR) which is less stringent than conventional one-copy-serializability.

In contrast to the above studies, our work retains the standard one-copy-serializability as the correctness criterion and focuses on the locking based concurrency control protocols. We also include an investigation of the O2PL algorithm which has not been studied before in the real-time context. The performance of the classical distributed 2PL locking protocol (augmented with the priority abort (PA) and priority inheritance (PI) conflict resolution mechanisms) and validation-based algorithms was studied in for realtime applications with "soft" deadlines operating on replicated DRTDBS[6].the performance of OCC is good only under light transaction loads. In [4], a conditional priority inheritance mechanism is proposed to handle priority inversion.

This mechanism capitalizes on the advantages of both priority abort and priority inheritance in real-time data conflict resolution. It outperforms both priority abort and priority inheritance when integrated with two phase locking in centralized real-time databases. However, the protocol assumes that the length (in terms of the number of data accesses) of transactions is known in advance which may not be practical in general, especially for distributed applications. In contrast, our

state-conscious priority blocking and state-conscious priority inheritance protocols resolve real-time data conflicts based on the states of transactions rather than their lengths.

In this section we will concentrate on the work that is under consideration for purposed work and we will also discuss some related replica concurrency control protocols. We review the 2PL [4] and O2PL [4] distributed CC protocols, and present our new CCRTRD protocol. All three protocols belong to the ROWA ("read one copy, write all copies") category with respect to their treatment of replicated data. In the following description, we assume that the reader is familiar with the standard concepts of distributed transaction execution [4, 8].

## III. CONCURRENCY CONTROL IN REAL-TIME DATABASE SYSTEMS

A Real-Time Database Systems (RTDBS) processes transaction A Real-Time Database Systems (RTDBS) processes transactions with timing constraints such as deadlines [58]. Its primary performance criterion is timeliness, not average response time or throughput. The scheduling of transactions is driven by priority order. Given these challenges, considerable research has recently been devoted to designing concurrency control methods for RTDBSs and to evaluating their performance. Most of these methods are based on one of the two basic concurrency control mechanisms: locking or optimistic concurrency control (OCC).

In real-time systems transactions are scheduled according to their priorities. Therefore, high priority transactions are executed before lower priority transactions. This is true only if a high priority transaction has some database operation ready for execution. If no operation from a higher priority transaction is ready for execution, then an operation from a lower priority transaction is allowed to execute its database operation.

Therefore, the operation of the higher priority transaction may conflict with the already executed operation of the lower priority transaction. In traditional methods a higher priority transaction must wait for the release of the resource. This is the priority inversion problem presented earlier. Therefore, data conflicts in concurrency control should also be based on transaction priorities or criticalness or both. Hence, numerous traditional concurrency control methods have been extended to the real-time database systems. In the following sections recent and related work in this area is presented.

### A. Phase Locking

In classical two-phase locking protocol, transactions set read locks on objects that they read, and these locks are later upgraded to write locks for the data objects that are updated. If a lock requested is denied, the requesting transaction is blocked until the lock is released. Read locks Call is shared. While write locks are exclusive. For real-time database Systems, two-phase locking needs to be augmented with a priority based conflict resolution scheme to ensure that higher priority transactions are not delayed by lower priority transactions. In High Priority scheme, all data conflicts are resolved in favour

of the transaction with the higher priority. When a transaction requests a lock on an object held by other transactions in a conflicting lock mode.

### B. Distributed Two Phase Locking

In this algorithm as described in [25] when a transaction arrives at a site in distributed system it is divided in sub transactions known as cohorts and processes which update replicas of data item are known as replica updaters. If a request by a cohort is a read lock on a data item then any lock is not required on the replicas at remote sites but if the request is a write lock then write locks are required on all the replicas of the dataitem. In this protocol write locks are obtained as the transaction executes, with the transaction blocking on a write request until all of the copies of the data items to be updated have been successfully locked by a local cohort and its remote updaters on replicas. Only the data locked by a local cohort is updated in the data processing phase of transaction. Remote copies locked by updaters are updated after those updaters have received list of items to be updated with the PREPARE message during the first phase of commit protocol. Write locks are only released after they are committed or aborted.

### C. Distributed Optimistic Two Phase Locking

There is another protocol as described in distributed environment for replica concurrency control as O2PL. In this algorithm when a cohort requests for a write lock, it is immediately given to it if lock is available. However it defers requesting write locks on replicas at remote site in the second phase of commit protocol.

In this protocol when a cohort updates a dataitem it requests for write locks on replicas after it has received PREPARE message from its master site. Actually what happens that after getting PREPARE message from its coordinator the cohort sends a PREPARE message to all of its remote updaters of the corresponding dataitem. With the PREPARE message it also sends list of the dataitems to be updated and the processes used in updating the dataitems. After that remote updaters obtain the locks on data item to be updated and sends COMMIT message to the cohort after completing the updation. Now after getting COMMIT message from replica updaters the cohort sends PREPARED message to its coordinator. Since the locks are deferred to the second phase of commit protocol there is a chance of both block and abort also due to arriving of higher priority transaction than the executing one.

Transactions with timing constraints such as deadlines [12]. Its primary performance criterion is timeliness, not average response time or throughput. The scheduling of transactions is driven by priority order. Given these challenges, considerable research has recently been devoted to designing concurrency control methods for RTDBSs and to evaluating their performance. Most of these methods are based on one of the two basic concurrency control mechanisms: locking or optimistic concurrency control (OCC).

In real-time systems transactions are scheduled according to their priorities. Therefore, high priority transactions are

executed before lower priority transactions. This is true only if a high priority transaction has some database operation ready for execution. If no operation from a higher priority transaction is ready for execution, then an operation from a lower priority transaction is allowed to execute its database operation.

Therefore, the operation of the higher priority transaction may conflict with the already executed operation of the lower priority transaction. In traditional methods a higher priority transaction must wait for the release of the resource. This is the priority inversion problem presented earlier. Therefore, data conflicts in concurrency control should also be based on transaction priorities or criticalness or both. Hence, numerous traditional concurrency control methods have been extended to the real-time database systems. In the following sections recent and related work in this area is presented.

## IV. THE CCRTRD PROTOCOL

We now present our new replica concurrency control protocol called CCRTRD augments. The O2PL protocol with a novel, simple to implement, state based conflict resolution mechanism called state-conscious priority blocking. In this scheme, the choice of conflict resolution method is a dynamic function of the states of the distributed transactions involved in the conflict. A feature of the design is that acquiring the state knowledge does not require inter-site communication or synchronization, nor does it require modifications of the two-phase commit protocol. Two real-time conflict resolution mechanisms are used in CCRTRD. Even though S2PL [4] is a deadlock free mechanism but it slows down the concurrent processing of multi transactions. This is due to locking of all the data till the end of the commit phase. Also if a higher transaction arrives at a site than executing one then current transaction is aborted and lock is made available to higher priority one. This makes the wastage resources. Hence we propose here a new mechanism with augmentation of S2PL. In classical two-phase locking protocol [14,15], transactions set read locks on objects that they read, and these locks are later upgraded to write locks for the data objects that are updated. If a lock requested is denied, the requesting transaction is blocked until the lock is released. Read locks can be shared, while write locks are exclusive. For real-time database systems, two-phase locking needs to be augmented with a priority-based conflict resolution scheme to ensure that higher priority transactions are not delayed by lower priority transactions. In High Priority scheme [13], all data conflicts are resolved in favor of the transaction with the higher priority. When a transaction requests a lock on an object held by other transactions in a conflicting lock mode, if the requester's priority is higher than that of all the lock holders, the holders are restarted and the requester is granted the lock; if the requester's priority is lower, it waits for the lock holders to release the lock. In addition, a new read lock requester can join a group of read lock holders only if its priority is higher than that of all waiting write lock operations. This protocol is referred to as 2PL-HP [13]. It is important to note that 2PL-HP loses some of the basic 2PL algorithm's blocking factor due to the partially restart-based nature of the High Priority scheme. We will use here a term High Priority point(HPP) with Block(BK)/ Donot Abort(DAB)

which means if a cohort reaches its High Priority point (HPP) than it will not be aborted against a higher priority transaction at that site. It means DAB is used here. And if a lower priority transaction demands a lock then it will be blocked against a higher priority executing one. It means BK is used here.

The proposed mechanism is:

- HPP of a cohort:

A cohort reaches its HPP after sending a PREPARE message to its replica updaters in its execution phase i.e. in first phase of 2PC.

- HPP of a replica updater:

A replica updater reaches HPP after gaining locks on needed data items. By this mechanism some significant improvements can be noted in S2PL. Since after HPP a cohort has a less probability of abortion hence a blocked transaction can borrow data from executing one. It means waiting and executing time of blocked transaction will get reduced and less probability to access which is very needed in a firm RTDBMS. Also by sending PREPARE message to its replica updaters as shown in the waiting time of a cohort between sending

PREPARE message to its updaters and receiving COMMIT message from them will get reduced. Hence over all time of execution of transaction will get reduced.

### Algorithm:

For a cohort:

HPP=F;

EXTfinish (execution finished) =f;

If (message=INITIATE COHORT)

{

Start execution of cohort;

EXTfinish=T;

}

If (EXTfinish=T)

{

Send PREPARE message to its replica updaters;

HPP=T;

HPP Send WORK DONE message to its coordinator ;

}

For a replica updater:

If (message=PREPARE && lock obtained=T)

{

HPP=T;

After execution send COMMIT message to its cohort;

}

## V. SIMULATION MODEL

To evaluate the performance of the concurrency control protocols described in above. We developed a detailed simulation model of a distributed real-time database system (DRTDBS). Our model is based on the distributed database model which has also been used in several other studies of distributed database system behavior, and the real-time processing model.

The database is modeled[17,18] as a collection of DBSize pages that are distributed over NumSites sites. Each page is fully replicated in our experiments. The physical resources at each site consist of NumCPUs CPUs, NumDataDisks data disks and NumLogDisks log disks. At each site, there is a single common queue for the CPUs and the scheduling policy is preemptive Highest- Priority-First. Each of the disks has its own queue and is scheduled according to a Head-Of-Line policy, with the request queue being ordered by transaction priority. The PageCPU and PageDisk parameters capture the CPU and disk processing times per data page, respectively. The parameter Init-WriteCPU models the CPU overhead associated with initiating a disk write for an updated page.

When a transaction makes a request for accessing a data page, the data page may be found in the buffer pool, or it may have to be accessed from the disk. The BufHitRatio parameter gives the probability of finding a requested page already resident in the buffer pool. The communication network is simply modeled as a switch that routes messages and the CPU overhead of message transfer is taken into account at both the sending and receiving sites and its value is determined by the MsgCPU parameter the network delays are subsumed in this parameter. This means that there are two classes of CPU requests – local data processing requests and message processing requests. We do not make any distinction, however, between these different types of requests and only ensure that all requests are served in priority order. With regard to logging costs, we explicitly model only forced log writes since they are done synchronously, i.e., operations of the transaction are suspended during the associated disk writing period. This logging cost is captured by the LogDisk parameter. Transactions arrive in a Poisson stream with rate Arrival Rate, and each transaction has an associated firm deadline, assigned as described below. Each transaction randomly chooses a site in the system to be the site where

the transaction originates and then forks off cohorts at all the sites where it has to access data. Transactions in a distributed system can execute in either sequential or parallel fashion. The distinction is that cohorts in a sequential transaction execute one after another, whereas cohorts in a parallel transactions are started together and execute independently until commit

Parameter	Meaning	Setting
NumSites	Number of sites	4
DBSize	Number of Pages in the databases	1000 pages
NumCPUs	Number of CPUs per site	2
NumDataDisks	Number of data disks per site	4
NumLogDisks	Number of log disks per site	1
BuffHitRatio	Buffer hit ratio on a site	0.1
ArrivalRate	Transactions arrived per second	Varied
SlackFactor	Slack factor in deadline assignment	6.0
TransSize	Av. No. of pages accessed per trans.	16 pages
UpdateProb	Update frequency per page access	0.25
PageCPU	CPU processing time per page	10 ms
InitWriteCPU	Time to initiate a disk write	2 ms
PageDisk	Disk access time per page	20 ms
LogDisk	Log force time	5 ms
MsgCPU	CPU message send/receive time	1 ms

TABLE I. SIMULATION MODEL PARAMETERS AND DEFAULT SETTINGS.

Processing is initiated. We consider only sequential transactions in this study. Note, however, that the execution of replica updaters belonging to the same cohort is always in parallel. The total number of pages accessed by a transaction, ignoring replicas, varies uniformly between 0.5 and 1.5 times TransSize. These pages are chosen uniformly (without replacement) from the entire database. The probability of accessed pages that are also updated is determined by UpdateProb. Upon arrival, each transaction T is assigned a firm completion deadline using the formula

$$\text{Deadline}_T = \text{ArrivalTime}_T + \text{SlackFactor} * \text{RT}$$

Where  $\text{Deadline}_T$ ,  $\text{ArrivalTime}_T$ , and  $\text{RT}$  are the deadline, arrival time, and resource time, respectively, of transaction T, while SlackFactor is a slack factor that provides control of the tightness/slackness of transaction deadlines. The resource time is the total service time at the resources at all sites that the transaction requires for its execution in the absence of data replication. This is done because the replica-related cost differs from one CC protocol to another. It is important to note that while transaction resource requirements are used in assigning transaction deadlines, the system itself lacks any knowledge of these requirements in our model since for many applications it is unrealistic to expect such knowledge. This also implies that a transaction is detected as being late only when it actually misses its deadline. As discussed earlier, transactions in an RTDBS are typically assigned priorities so as to minimize the number of killed transactions. In our model, all cohorts inherit their parent transaction's priority. Messages also retain their sending transaction's priority. The transaction priority assignment used in all of the experiments described here is the widely-used Earliest Deadline First policy, wherein transactions with earlier deadlines have higher priority than transactions with later deadlines. Deadlock is possible with some of the CC protocols that we evaluate – in our experiments; deadlocks are detected using a time out mechanism.

All the MissPercent graphs in this paper show mean values that have relative half widths about the mean of less than 10% at the

90% confidence interval, with each experiment having been run until at least 10000 transactions are processed by the system. Only statistically significant differences are discussed here.

## VI. EXPERIMENTS AND RESULTS

The experiment has to be performing using different simulation language like C++ Met lab etc. Table 1 presents the setting of the simulation model parameters for our experiment. With these settings, the database is fully replicated and each transaction executes in a sequential fashion. The performance metric employed is MissPercent, the percentage of transactions that miss their deadlines. MissPercent values in the range of 0 to 25 percent are taken to represent system performance under “normal” loads, while MissPercent values in the range of 30 to 100 percent represent system performance under “heavy” loads. Several additional statistics are used to aid in the analysis of the experimental results, including the abort ratio, which is the average number of aborts per transaction, the message ratio, which is the average number of messages sent per transaction, the priority inversion ratio (PIR), which is the average number of priority inversions per transaction, and the wait ratio.

Our goal in this experiment was to investigate the performance of the various conflict resolution mechanisms. For this experiment, Figures 1 and 2 present the MissPercent behavior of the O2PL, MIRROR and CCRTRD protocols under normal loads and heavy loads, respectively. To help isolate the performance degradation arising out of concurrency control.

We observe that O2PL and MIRROR have similar at arrival rates lower than 9 transactions per second, but MIRROR outperforms O2PL under heavier loads. This is because O2PL-PA ensures that urgent transactions with tight deadline scan proceed speedily since they are not made to wait for transactions with later deadlines in the event of data conflicts. From collected statistics, we have found out that O2PL greatly reduces the priority inversion ratio, the wait- ratio and the wait time as compared to MIRROR.

This is due to the inherent non-preempt ability of prepared data of a low priority cohort (updater) which has already reached the PREPARED state at the time of the data conflict. In this case, it cannot be aborted unilaterally since its destiny can only be decided by its master and therefore the conflicting high priority transaction is forced to wait for the commit processing to be completed.

Our attention to the CCRTRD protocol, we observe that CCRTRD has the best performance among all the protocols. The improved behavior here is due to CCRTRD's feature of avoidance of transaction abort after a cohort (or updater) has reached its demarcation point. The importance of this is quantified in Figure3&4, in such situations, a high priority transaction may afford to wait for a lower priority transaction to commit since it is near completion, and wasted resources due to transaction abort can be reduced, as is done by the CCRTRD protocol. In fact, CCRTRD does better than all the other O2PL based algorithms under all the workload ranges.

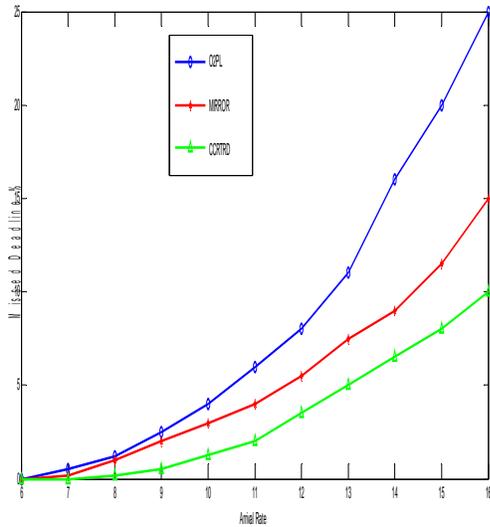


Figure 1. Normal Load

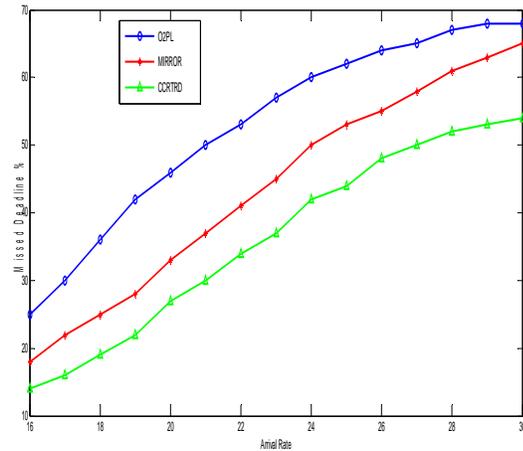


Figure 2. Heavy Load

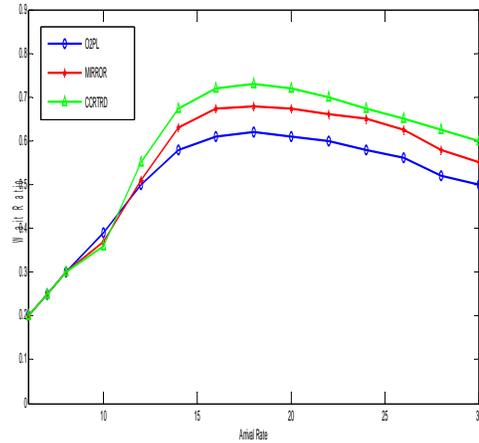


Figure 3. Wait Ratio

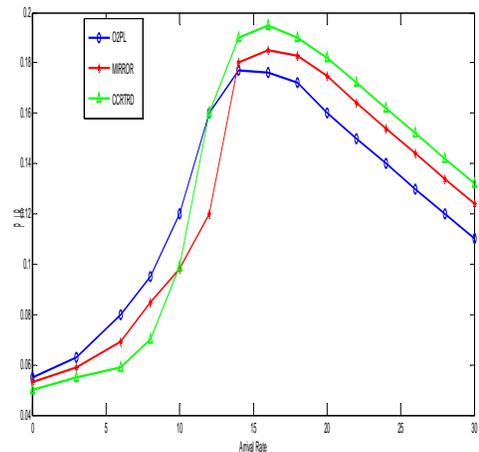


Figure 4. Demarcation Point

## Figure-O2PL Based Protocol

### VII. PROFICIENCY OF CCRTD

We have addressed the problem of managing firm-real time database system that access replicated data in a distributed system that may be replicated. For this environment in which many current time-critical applications operate, especially Web-based services, we proposed a CC protocol called CCRTD that can be easily integrated in current systems to handle processors without altering the database consistency which is the main objective of DRTDBS. The main idea of the protocol is to associate an importance value to each submitted transaction in order to find out as

- By this proposed mechanism a deadlock free environment is provided.
- The waiting time of blocked transaction is reduced.
- Also the execution time of current transaction is reduced.
- Wasting of resources is minimized.
- Easy implementation with integration of S2PL and 2PC.
- Decreases the waiting time of transactions in wait queue.

### VIII. CONCLUSIONS

At this point we have in exploit the difficulty of access replicated data in firm RTDBS mainly concurrency control in real time replicated database problem in conflict mode. This proposed mechanism can be easily integrated and implemented in current systems. However if we talk about advantages of this proposed CCRTD protocol over other protocols like O2PL, and MIRROR then this mechanism will lead in some areas like probability of deadlock is decreased. Also by sending message, the waiting time of cohort after sending PREPARE message to its updaters and receiving COMMIT message from them is decreased than in MIRROR due to which whole time of Execution phase and Commit phase will get decreased. Also in this paper it is said that a blocked transaction can borrow data item after reaching the High Priority point by the executing transaction, in this way the waiting time of transaction in queue will get decreased.

Hence from above discussion we recommend this CCRTD Protocol over other protocols in replicated environment of firm RTDBS.

### REFERENCES

- [1] Gray, J., "Notes On Database Operating Systems," in Operating Systems: An Advanced Course, R. Bayer, R. Graham, and G. Seegmuller, eds., Springer-Verlag, 1979.
- [2] M. Valduriez P, 1991, Principles of Distributed Database Systems, Prentice-Hall. P. Bernstein, V. Hadzilacos and N. Goodman,
- [3] M. Xiong et al. "MIRROR: A State-Conscious Concurrency Control Protocol for Replicated Real-Time Databases", Computer Science Department Faculty Publication Series, 1999
- [4] U. Shanker et al., "The SWIFT-Real Time Commit Protocol", International Journal of Distributed and Parallel Databases, Springer Verlag, Vol. 20, Issue 1, 2006, pp. 29-56
- [5] El Abbadi, A., Toueg, S., 1989. Maintaining availability I partitioned replicated databases. ACM Trans. Database Syst. 14 (2), 264-290.
- [6] Ramamritham, Son S. H, and DiPippo L, 2004, Real-Time Databases and Data Services, Real-Time Systems J., vol. 28, 179-216.
- [7] Robert A and Garcia-Molina H, 1992, Scheduling Real-Time Transactions, ACM Trans. on Database Systems, 17(3).
- [8] Levy E., Korth H and Silberschatz, 1991, An optimistic commit protocol for distributed transaction management, Pro. of ACM SIGMOD Conf.
- [9] Jayant. H, Carey M, Livney, 1992, "Data Access Scheduling in Firm Real time Database Systems", Real Timesystems Journal, 4(3)
- [10] Jayanta Singh and S.C Mehrotra, 2006, "Performance analysis of a Real Time Distributed Database System through simulation" 15th IASTED International Conf. on APPLIED SIMULATION & MODELLING, Greece
- [11] Jayanta Singh and S.C Mehrotra, 2009 "A study on transaction scheduling in a real-time distributed system", EUROESIS's Annual Industrial Simulation Conference, UK.
- [12] Ramamritham, Son S. H, and DiPippo L, 2004, Real-Time Databases and Data Services, Real-Time Systems J., vol. 28, 179-216.
- [13] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," Proceedings of the 14th VLDB Conference, August 1988.
- [14] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data," Proceedings of the 15th VLDB Conference, August 1989.
- [15] Eswaran, K. P., J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," Communications of the ACM, 19(11), November 1976.
- [16] Son, S., "Using Replication for High Performance Database Support in Distributed Real-Time Systems," Proceedings of the 8th IEEE Real-Time Systems Symposium, pp. 79-86, 1987
- [17] M Xiong, Sivasankaran, R., Stankovic, J.A., Ramamritham, K. and Towsley, D., "Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics," Proceedings of the 17th IEEE Real-Time Systems Symposium, pp. 240-251, Washington, DC, December 1996
- [18] M. Xiong et al. "MIRROR: A State-Conscious Concurrency Control Protocol for Replicated Real-Time Databases", Computer Science Department Faculty Publication Series, 1999

### Authors

#### Ashish Srivastava:



Received his Master degree in Computer Application in year 2002 & M.Sc (Math) in year 2004 presently he is working as Sr.Lecturer, M.B.S.P.G. College, Gangapur, Varanasi. He has more than 12 years teaching and professional experience. His area of interest

includes DBMS, Discrete Mathematics Distributed Real Time Database Systems & Networks. He is pursuing his Ph.D. in Computer Science on "Issues in Distributed Real Time Replicated Database System". He has published several papers in National and International conferences and Journals.

#### Dr. Udai Shankar:



Received his Master degree in Computer Science in year 1998 and Ph.D. degree in 2006 from IIT, Roorkee (U.P.). Presently he is working as Professor in Computer Science and Engineering Department, Madan Mohan Malaviya university of Technology, Gorakhpur. He has more than 25 years teaching experience. His areas of interest include Database Systems, Real Time Systems, Distributed Real Time Database Systems Mobile Distributed Real Time Database Systems and Grid Databases. He has published more than 30 papers in National and International conferences and Journals.