A Database Replication Protocol for Distributed Real Time System

Sanjay Kumar Tiwari[1]
Deptt. of Computer Sc. & Engineering
Madan Mohan Malaviya Engineering College, Gorakhpur. India

A.K.Sharma[2]
Deptt. of Computer Sc. & Engineering
Madan Mohan Malaviya Engineering College, Gorakhpur, India

Vishnu Swaroop[3]
Deptt. of Computer Sc. & Engineering
Madan Mohan Malaviya Engineering College, Gorakhpur, India

Abstract— **Database replication protocols based on a certification approach are usually the best ones for achieving good performance. The weak voting approach achieves a slightly longer transaction completion time, but with a lower abortion rate. So, both techniques can be considered as the best ones for replication when performance is a must, and both of them take advantage of the properties provided by atomic broadcast. The propose a new database replication protocol that shares many characteristics with such previous strategies. It is also based on totally ordering the application of writesets, using only an unordered reliable broadcast, instead of an atomic broadcast. Additionally, the writesets of transactions that are aborted in the final validation phase are not broadcast in our strategy modified protocol is suited for real time application. Thus, this new approach reduces the communication traffic and also achieves a good transaction response time.**

Keywords- Real time system, replicated database, certification protocol, deterministic protocol, weak-voting protocol, validate transaction, DRTDBS.

I. INTRODUCTION

Real-time systems can be defined as those computing systems that are designed to operate in a timely manner. That is, performing certain actions within specific timing constraints; e.g., producing results while meeting predefined deadlines. Hence, the notion of correctness of a real-time system is contingent upon the logical correctness of the produced results as well as the timing at which such results are produced. [1, 2]. Database replication based on group communication systems has been proposed as an efficient and flexible solution for data replication. Protocols based on group communication typically rely on a broadcast primitive called atomic [3] or total order [4] broadcast. This primitive ensures that messages are delivered reliably and in the same order on all replicas. This approach ensures consistency and increases availability by relying on the communication properties assured by the total order broadcast primitive. This primitive simplifies greatly the development of replication protocols: prevents the usage of an atomic commitment protocol [5], avoid the occurrence of distributed deadlock cycles and offer a constant interaction for committing a transaction. A comparison of database replication techniques based on total order broadcast is introduced in [6]. From those presented there, the two leading techniques, in performance terms, are: certification-based [7] and weak-voting [8] protocols.

Certification-based protocols- Certification-based protocols keep at each replica an ordered log of already committed transactions. When a transaction requests its commit the writeset1 is multicast in a message, using the total-order service. Messages are treated by the replication protocol in the same order in which they are delivered at the replicas. Each writeset is certified against the information contained in the log, according to some predefined rules that depend on the required isolation level [9], in order to abort or commit the delivered transaction. In the former case, the writeset is discarded (except for its the delegate replica, where the transaction gets aborted) and, in the latter case, the writeset will be applied and committed at the remote replicas while it will be straightly committed at its delegate replica.

Weak-Voting Protocols- The weak-voting protocols also send the writeset using the total-order multicast upon the commit request of a transaction. When a writeset is delivered to a replica, it is atomically applied so it may cause the abortion of other existing local transactions. If an aborted transaction had already sent its writeset the protocol would multicast an abort message (using a weaker multicast delivery service [4]) to notify that the transaction must be aborted at all the replicas. When the writeset is delivered at its delegate replica and the transaction remains active (i.e. no other previous delivered writeset has rolled it back), the transaction will be committed and an additional message will be multicast to commit the transaction at the rest of the replicas . From the previous descriptions, it should be clear that certification-based protocols need just one total-order message round per transaction whereas weak-voting ones need an additional round. Thus, the first ones present a better behavior in terms of performance but higher abortion rates, since transactions may stay too long in the log (raising conflicts with new transactions being certificated) until removed from it after having committed in all replicas. Recently, Database Management Systems (DBMS) providing Snapshot Isolation (SI) [10] have become widely used; since this isolation level allows that read-only transactions are never blocked, using multi-version concurrency control. In this way, several certification-based protocols have been proposed to achieve this isolation level in a replicated setting [9, 11, 12, 13], whilst quite a few weak-voting ones [14, 15]. As it is well-known, database replication

ensures higher availability and performance of accessed data. Hence, it is important to study other alternatives to the already presented replication protocols; unfortunately, it is quite difficult to find an alternative to those based on the total order multicast. From our point of view, total-order based replication protocols offer two key properties: they reliably send the writeset to all replicas; and they provide the same scheduling of transactions and hence all replicas reach the same decision for each transaction submitted to the replicated system.

This protocol [16] follows at each replica the most straightforward scheduling policy: at a given slot, only those writesets coming from a given replica are allowed to commit; other conflicting local transactions should be aborted to permit those writesets to commit. Actually, this is a round-robin policy based on replica identifier which is unique and known by all the nodes of the system (since all replicas may know the identifiers of the other ones). In this deterministic protocol, a transaction is firstly executed at its delegate replica and once it requests for its commit, its updates are stored in a data structure and it will be committed when the turn of its delegate replica arrives (at its corresponding slot). Then, the replica will multicast all writesets from transactions that requested their commit since the last slot and they will be sequentially applied at the rest of replicas (after all writesets from the previous slot have been applied). Hence, it is easy to show that all local conflicting transactions are aborted and only those that survived will be multicast in their appropriate slot.

This generates a unique scheduling configuration of all replicas, in which all writesets are applied in the same order at all replicas. If we assume that the underlying DBMS at each replica provides SI the deterministic protocol will provide Generalized SI (GSI). The atomicity and the same order of applying transactions in the system have been proved in [17] to be sufficient conditions for providing GSI. We provide some discussion about this fact in this paper. This new approach provides several advantages over the previously presented techniques. Compared to the certification-based protocols, our approach does not use a certification log with the writesets of committed transactions and therefore there is no need of using a garbage collector to avoid its boundless growing. Compared to the weak voting protocols, it avoids the second message round to confirm the outcome of the transaction, since all multicast writesets are going always to commit. For the same reason, this approach reduces the network traffic and also the resource consumption of the replicas. A replica will never multicast writeset that finally aborts, avoiding its unnecessary delivery through the network and processing at the remote replicas. We have simulated a scenario to compare this approach with a typical distributed certification protocol and we have verify that the abortion rate is reduced in many cases while maintaining very similar response times. Finally, we provide some outlines about how to deal with fault-tolerance issues, such as the failure of a replica and its subsequent re-join to the system.

II. SYSTEM MODEL

On behalf of our protocol proposal, we take advantage of the capabilities provided by a middleware architecture called MADIS [13]. Users and applications submit transactions to the system. The middleware forwards them to the respectively nearest (local) replica for their execution, i.e. its delegate replica; the way it is chosen is totally transparent to the behavior of the protocol. We assume a partially synchronous distributed system where message propagation time is unknown but eventually bounded. The system is composed by N replicas (R0, . . . ,RN−1) and each one of them holds a complete copy of a given database, i.e. full replication. An instance of the deterministic protocol is running in each replica and runs on top of a DBMS that provides SI.

A replica interacts with other replicas thanks to a Group Communication System [4] (GCS) that provides a reliable multicast communication service without any other ordering assumption rather than the reliable delivery of messages despite failures. Besides, the GCS also provides a membership service, which monitors the set of participating replicas and provides them with consistent notifications in case of failures, either real or suspected.

III. DETERMINISTC PROTOCOL

This Section describes the operation of the modified Determ-Rep protocol algorithm [18] for Replication in Real Time System executed by the middleware at a replica Rk (Figure 3.2), considering a fault-free environment. Details about the failure and rejoin of a replica will be depicted in Section 5. All operations of a transaction T are submitted to the middleware of its delegate replica (explicit abort operations from clients are ignored for simplicity). At each replica, the middleware keeps an array (to work) that determines the same scheduling of transactions in the system. Here, for the sake of understanding, it is assumed a round robin scheduling based on replica identifiers. In other words, to work is in charge of deciding which replica is allowed to send a message, or which writesets have to be applied and in which order. Each element of the array represents the actions that have to be performed when their turn arrives and each one is processed cyclically according to the work_turn.The middleware forwards all the operations but the commit operation to the local database replica (step I of Figure 3.2). Each replicamaintains a list (tocommit_wslist) which stores local transactions (T.replica = Rk) that have requested their commit. Thus, when a transaction requests its commit, the writeset is retrieved from the local database replica (T.WS). If it is empty the transaction will be committed straight away, otherwise the transaction (together with its writeset) will be stored in tocommit_ wslist.

In order to commit transactions that have requested it, the replica has to multicast the stored writesets in a tocommit message and then wait for the reception of this message to finally commit the transactions. Since our protocol follows a round robin scheduling, the replica hast to wait for its turn (work turn = Rk in step III), so as to multicast all the writesets contained in tocommit wslist using a simple reliable service. When the turn of a replica arrives and there are no transactions

stored in tocommit wslist, the replica will simply advance the turn to the next replica, sending a next message to all the replicas.

Upon delivery of any of these messages (next and tocommit) at each replica, they are stored in their corresponding positions in the towork array, according to the site Rk which the message came from (step II). It is important to note that, although these messages were sent since it was its turn at its replica, all replicas run at different speed and there can be replicas still handling previous positions of their own to work. Messages from different sites may be delivered disordered (as we do not use total order), but this is not a problem since they are processed one after another as its turn arrives. Disordered messages are stored in their corresponding positions in the array and their processing will wait for the delivery and processing of the previous ones.

This ensures that all the replicas process messages in the same order and as a result all transactions are committed in the same order in all of them. Thus, the towork array is processed in a cyclical way. At each turn, the protocol checks the corresponding position of the array (towork[work_ turn]). If a next message is stored, the protocol will simply remove it from the array and change the turn to the following position (step IV) so as to allow the next position to be processed. If it is a tocommit message, we can distinguish between two cases (step V). If the sender of the message is the replica itself, transactions grouped in its writeset are local (already exist in the local DBMS) and therefore the transactions will be straightforwardly committed. In the other case, a remote transaction has to be used to apply and commit the transaction.

In this case, special attention must be paid to local existing transactions since they may conflict with the remote writeset application, avoiding it to progress. To partially avoid this we stop the execution of write operations in the system (see step I.2.a in Figure 3.2) when a remote writeset is applied at a replica, i.e. turning the ws _run variable to true. However, this is not enough to ensure the writeset application in the replica; the writeset can be involved in a conflict with local transactions that already updated some data items that intersect with the writeset. This is ensured by a block detection mechanism, presented in [13], which aborts all local conflicting transactions (VI) allowing the writeset application to be successfully applied. Besides, this mechanism prevents local transactions that have requested their commit (T.precommit = true) from being aborted by other local conflicting transactions, ensuring their completion. Note also that the writeset application may be involved in deadlock situation that may result in its abortion and hence it must be re-attempted until its successful completion.

A. Prediction of Time-Constraint Violation for Real Time System

The Real Time System predicts deadline violation of tasks based on a feasibility test that takes into account the scheduling time of a phase, as well as the current time, deadline, and processing time of tasks. Accounting for the scheduling time in the feasibility test ensures that no task will miss its deadline due to scheduling time. The test for adding a task assignment (Tl Pk) to the current feasible partial schedule CPS to obtain feasible partial schedule CPS' in scheduling phase j is performed as shown in Table 3. 1.

IF (tc+RQs(j) + selk ≤ dl)
THEN CPS' is feasible
ELSE CPS' is infeasible
Abort (T)

Table.3.1. Feasibility test of Real Time System

In the above feasibility test, tc indicates the current time, RQs(j) indicates the remaining time of scheduling i.e. RQs = Qs - (tc - ts), where Qs is the allocated scheduling time to phase j, and ts is the scheduling start-time. Finally, sel_k is the scheduled end time for task T_l on processor Pk.

Next, we provide a theorem that guarantees to minimize to 0 the number of scheduled tasks whose deadlines are missed during their execution.

Theorem: The tasks scheduled by RTDS (Real Time Dynamic System) are guaranteed to meet their deadlines, once executed.

Proof: The proof is done by contradiction. Let us assume that a task $T_l \in$ Batch(j) is scheduled on processor Pk during the jth phase and that T_l misses its deadline at execution time. This assumption leads to the following condition:

$$te(j) + sel_k > dl \quad (1).$$

Here we are assuming that the execution of tasks in a schedule will start immediately after scheduling ends. On the other hand, RTDS's bound on scheduling-time allocated to each phase ensures that:

$$te(j) \leq tc + RQs(j) \quad (2).$$

Combining (1) and (2) leads to:

$$tc + RQs(j) + sel_k > dl \quad (3).$$

The feasibility test performed at time tc ensures that: $tc+RQs(j)+sel_k \leq dl$, contradicting inequality [19].Therefore, our assumption regarding deadline violation of Tl is false, which concludes the proof of the theorem.

B. Modified Algorithm for Replication in Real Time System

Initialization:

1. ws_ run := false
2. tocommit_ wslist := ∅
3. towork[i] := ∅ with i ∈ 0..N-1
4. work turn := 0

Step-I. Upon operation request for T from local client for a constant of time

IF (tc+RQs(j) + selk ≤ dl

THEN CPS' is feasible

ELSE CPS' is infeasible

Abort (T)

/* Transactions Abort after the constant of time */

1. if SELECT then

a. execute operation at Rk and return to client

2. else if UPDATE, INSERT, DELETE then

a. if ws_ run = true then

- wait until ws_ run = false

b. execute operation at Rk and return to client

3. else if COMMIT then

a. T.WS := getWriteset(Tj) from local Rk

b. if T.WS = ∅ then

- commit T and return to client

c. else

- T.replica := Rk

- T.pre commit := true

- tocommit_ wslist := tocommit _wslist .{ T }

Step-II. Upon receiving message msg

/* .{next,Rn} or {tocommit,Rn, seq _txns} */

1.Store msg in towork[Rn]

Step-III. Upon replica's turn /* work_ turn = Rk */

1.if tocommit _wslist = ∅ then R_multicast(.{next, Rk })

2.else R_multicast({tocommit, Rk, tocommit_ wslist})

Step-IV. Upon {next, Rn} in towork[work_ turn]

1.Remove {next, Rn} from towork[work _turn]

2.work _turn := (work _turn+1) mod N

Step-V. Upon {tocommit, Rn, seq _txns} in towork[work _turn]

1. while seq_ txns ≠ ∅ do

a. T' := first in seq _txns

b. if T'.replica = Rk then /* T' is local */

- commit T' and return to client

c. else /* T' is remote */

- ws_ run := true

- apply T' .WS to local Rk

/* T' may be reattempted */

- commit T'

2. Remove .{tocommit, Rn, ∅ }from towork[work _turn]

3. work_ turn := (work _turn+1) mod N

4. ws_ run := false

Step-VI. Upon block detected between T1 and T2

/* T1.replica ≠ Rk */

/* T2.replica = Rk, i.e. local */

1. abort T2 and return to client

2. if T2.pre commit = true then

a. remove {T2} from tocommit _wslist .

FIG. 3.2. ***Algorithm for Replication in Real Time System***

C. Protocol Optimizations

Several optimizations can be considered for this protocol. If we take a look on how writesets are applied at remote replicas, there can be several alternatives. In Figure 3.2, we have chosen to submit the writesets one by one into the local database. However, they can be grouped in a single remote transaction, reducing the system overhead and therefore increasing the system performance.Another important enhancement permits reducing the time a replica must spend to multicast a message when its turn arrives. Upon receiving a tocommit message, it is possible to know in advance which local committing transaction (stored in tocommit wslist) are going to abort and which not. Therefore, if it is the turn of the replica, it may gather the transactions that are not going to abort and send the corresponding message without waiting for the writeset application in the local database replica. This

optimization has been considered when it comes to performing our tests.

Finally, we can also consider a scheme of transactions based on conflict classes [20 21] in order to increase the system performance. A conflict class represents a partition of the data and each application may partition the data depending on its requirements (e.g. there could be a class per table). If these partitions are well chosen, increasing the protocol concurrency may be possible and hence the system performance. When it comes to work with conflict classes, it is possible to avoid blocking all transaction operations in the local database in order to apply a remote writeset. In this case, existing transactions belonging to the same conflict class of the writeset are aborted in order to guarantee its successful application. However, it is only necessary to block operations from transactions of the same conflict class, while allowing the others to progress normally.

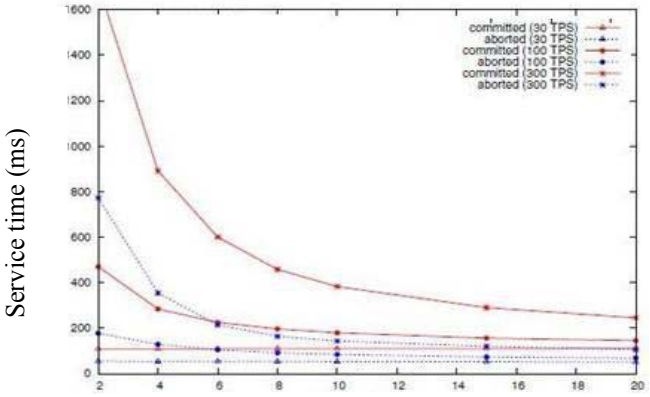
IV. CORRECTNESS DISCUSSION

In this Section we outline the discussion about the correctness of our replication protocol. In the following, we assume that we are under a failure-free environment. First of all, we have to show that every writeset submitted to a database will be eventually committed.

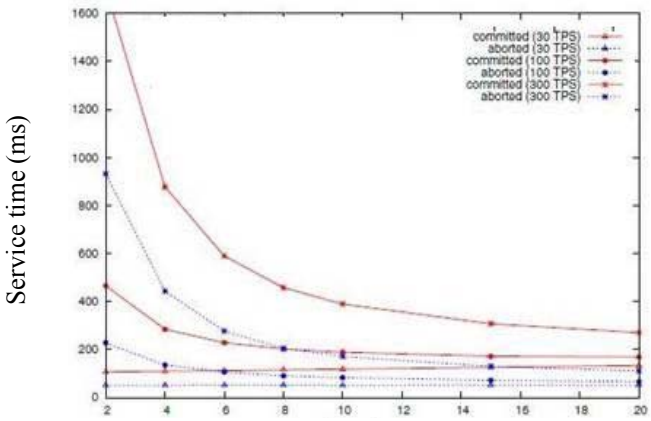
Theorem 1. Given a transaction Ti $\in$ T, whose delegate replica isRk with k $\in$ N, then its associated multicast writeset (Ti.WS) will be eventually committed.

Proof. We have to distinguish whether it is executed at Rk or at a remote one Rj with j ≠ k. In the first case, it will be committed as soon as the reliable message is delivered (see Figure 3.2, step V) since it already acquired all items it has to update. While in Rj , its associated tocommit message has to be delivered and scheduled (i.e. the turn in Rj reaches the position of the message in towork, which corresponds with its delegated replica).

At that moment, it is submitted to the database (at the same time none local transactions are allowed to write anymore nor any other remote writeset is scheduled) and thanks to the block detection mechanism between transactions all possible local conflicting transactions will be eventually rolled back and, since no new write operations are allowed but the ones issued by Ti, the Ti.WS will be applied and committed.



a-Certification-based Protocol



b-Deterministic Protocol

FIG.3.3.TRANSACTION COMPLETION TIME WITH 0% READ ONLY (RO).

In the following we proof the atomicity of transactions; informally, if a transaction is committed at a given replica, it will be eventually committed at all replicas.

Theorem 2 (Atomicity of transactions). If a (tocommit,Rn,seq_txns) is processed and committed at a replica Rk with k $\in$ N, then it will be eventually processed and committed at all replicas.

Proof. This proof can be split into several parts. Let us denote as Rk' with k'$\in$N $\wedge$ k ≠ k' the replica to analyze. Reception of {tocommit,Rn,seq_txns} at Rk'. The message will be received since it has been received by Rk due to the fact that it was multicast by its associated delegate replica (in this case Rn) using the reliable channels between replicas. The {tocommit,Rn,seq_ txns} message reaches its turn in towork at Rk' (i.e. work_turn = Rn). First of all, it is worth noting that replica Rk' may run slower (if it is faster, it will be the other way round, exchanging replica identifiers) and its respective work turn may be different and hence Rk may have already processed some items. Let us denote as n the position of the message { tocommit,Rn, seq_txns } in towork at Rk' . Thus, we must ensure that the distance between n and work_ turn is decreased and hence the message will be processed (i.e. writesets contained in seq_ txns will be applied and committed). If we consider that the current position (work_ turn) of towork is a next message, it will be removed from towork and the turn will advance to the next position; hence, the distance shortened. Otherwise, it is a tocommit message and its associated writesets will be eventually committed, by Theorem 1, and they will be removed from towork and the turn will advance to the next position; hence, the distance again decreased. Therefore, the turn will eventually reach the position of the message in the towork array.

The {tocommit,Rn,seq txns } is processed at Rk'. This is easily shown by Theorem 1. Once the turn reaches the position of the message, the writesets will be successfully applied in

the database. The atomicity of transaction does not ensure that all transactions are committed in the same order. If we ensure that all transactions are committed in the same order at all replicas then it will be satisfied a sufficient condition for generating GSI histories [9].

Theorem 3 (Same commit order at all replicas). All terminated transactions should follow the same commit order in all replicas.

Proof. Due to Theorems 1 and 2 we know that a transaction committed at a replica will be committed at all replicas. It is easy to show that they will be applied in the same order thanks to the way towork is built. When different next or tocommit messages are delivered they are inserted at their appropriate towork slots. Writesets are extracted and applied in the cyclical order they are located in towork and they are committed in the same order they are applied. Hence, it is ensured that all replicas commit the same set of transactions in the same order.

V. SIMULATION ANALYSIS

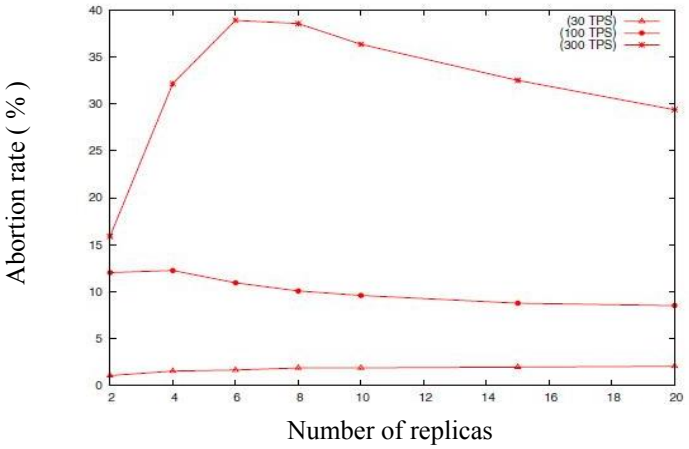
We have compared our deterministic protocol with a certification-based protocol [15]. The simulation parameters have been summarized as this table shows; we have chosen a slow local network. This provides the worst results for our deterministic protocol since it depends a lot on the network delays. In practice, the combination of a reliable broadcast and a turn-based sending privilege can be considered as a way of implementing a total-order broadcast, as already discussed above.

S.NO.	PARAMETER	VALUES
1.	Replicas	2 to 20
2.	Global TPS load	30, 100, 300
3.	Database size	10000 items
4.	Item size	200 bytes
5.	Mean WS size	15 items
6.	Mean RS size	15 items
7.	Min.Trans.length	100 ms
8.	WS application time	30 ms
9.	Connections	6/replica
10.	Message delay	3 ms
11.	Trans.per test	40000
12.	Read-only trans	0%, 80%
13.	Dead line of Execution time	Ti

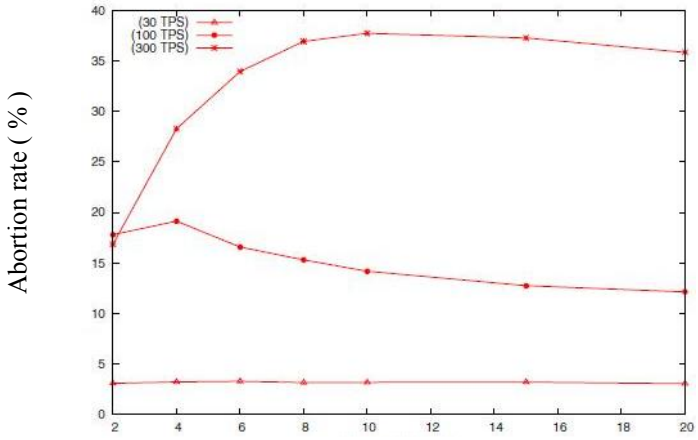
Table.5.1. Simulation parameters.

Each transaction consists of an update and a reading sentence. For read-only transactions, the update has been replaced by another reading sentence. Each operation involves 15 items on average. The minimal transaction length is set to 100 ms, adding an inter-sentence delay that ensures that the transaction time is at least 100 ms.

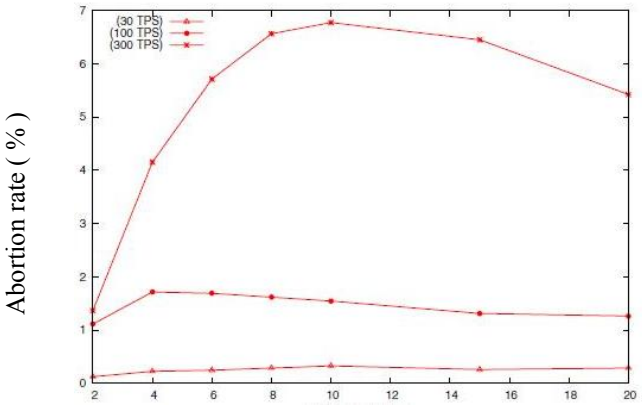
In our tests, we have used 6 connections per node, checking the protocols behavior in a LAN environment and using a worst case load consisting only in read-write transactions and another with 80% of read-only transactions (the common case in many real applications). Besides the transaction completion time, we also analyze their abortion rate in both cases. Simulation (Fig.3.3.a and 3.3.b) shows that the results in the worst case, i.e. when no read-only transactions are included in the simulated load. Completion times for committed transactions are the same when the system consists of less than 10 replicas. Bigger systems generate slightly longer times in the deterministic approach. On the other hand, the certification strategy is able to abort transactions earlier, although the differences are only significant with the highest simulated load (300 TPS). Response times for other tested loads follow the same trend, i.e. there are no significant differences between both protocols. Comparing both protocols, with an update-only load (Fig.3.4.a and 3.4.b) the deterministic one provides the best results for light and medium loads (30, 100 TPS), always at least 20% better than the certification-based protocol and in some cases more than 50% better. However,



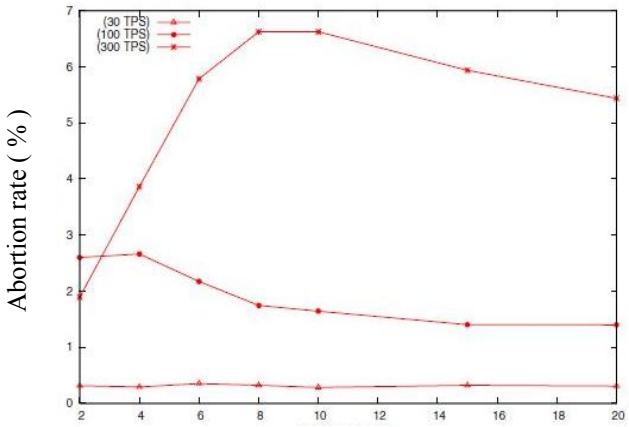
a-Deterministic (0% Read only).



b-Certification based (0% Read



c-Deterministic (80% Read only).



d-Certification based (80% Read only).

Fig.3.4.Abortion Rates.

in the heaviest load case (300 TPS) things are not so clear. The deterministic protocol has its maximum value when 6 replicas are used, whilst the certification-based protocol has its maximum with 10 replicas. Due to this, the deterministic protocol is better than the certification-based one when the system has more than 8 replicas. Simulation results (Fig.3.4.c and 3.4.d) shown that the abortion rates when 80% of the transactions are read-only. In general, the results show similar trends to the previous case, i.e. with light and medium loads the deterministic protocol is much better than the certification-based one. With the heaviest load no clear winner can be identified, as it already occurred before, but now the curves follow different trends. When there is no change (write/update) operation in transaction i.e. read only transaction need not require replication and service time of transaction will decrease to null but if 0% Read only that is any change is required in transactions for the replication. Number of TPS increases and service time is counted. The number of TPS increases in that ratio service time increased and abortion also increase. But in real time system the abortion is a big issue for any system. So number of TPS should be limited to any real time system. Research Studies show that the deterministic protocol is better than certification based protocol for the real time system. Because abortion rate is less than the certification based protocol in compare to deterministic protocol.

VI. CONCLUSIONS

The deterministic database replication protocol for Distributed Real Time Database System is able to inherit the best characteristics of both certification based as well as may be for weak-voting approaches. Thus, like a weak voting protocol, it is able to validate transactions without logging history of previously delivered writesets, and like a certification-based protocol, it is able to validate transactions using only a single round of messages per transaction. Moreover, such a single round can be shared by a group of transactions already served at the same delegate replica. The correctness of this new protocol for DRTDBS has been justified. Additionally, its performance has been analyzed through simulation, providing a transaction completion time quite similar to that of a certification-based approach (the best one according to previous analysis [6]) in some configurations, and with a lower abortion rate.

REFERENCES

[1] F. Panzieri, R. Davoli, “Real Time Systems: A Tutorial”, Technical Report UBLCS-93-22, ftp://ftp.cs.unibo.it/pub/techreports, University of Bologna, Bologna (Italy).

[2] John A. Stankovic, “On Real-Time Transactions”, SIGMOD Record, Vol. 17, No. 1, March 1988.

[3] V. Hadzilacos and S. Toueg. A modular approach to faulttolerant broadcasts and related problems. Technical Report TR94-1425, Dep. of Computer Science, Cornell University, Ithaca, New York (USA), 1994.

[4] P. A. Bernstein, D.W. Shipman, and J. B. R. Jr. Concurrency control in a system for distributed databases (sdd-1). ACM Trans. Database Syst., 5(1):18–51, 1980.

[5] P. A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison- Wesley, 1987.

[6] M. Wiesmann and A. Schiper. Comparison of database replication techniques based on total order broadcast. IEEE TKDE, 17(4):551–566, 2005.

[7] F. Pedone. The database state machine and group communication issues (These N. 2090). PhD thesis, Ecole Polytecnique Federale de Lausanne, Lausanne, Switzerland, 1999.

[8] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. ACM Trans. Database Syst., 25(3):333–379, 2000.

[9] S. Elnikety, F. Pedone, and W. Zwaenopoel. Database replication using generalized snapshot isolation. In SRDS. IEEECS, 2005.

[10] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O’Neil, and P. O’Neil. A critique of ANSI SQL isolation levels. In SIGMOD, 1995.

[11] Y. Lin, B. Kemme, M. Patino-Mart´ınez, and R. Jimenez- Peris. Middleware based data replication providing snapshot isolation. In SIGMOD. ACM, 2005.

[12] S. Wu and B. Kemme. Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation. In ICDE, pages 422–433. IEEE-CS, 2005.

[13] F. D. Munoz-Escoı, J. Pla-Civera, M. I. Ruiz-Fuertes, L. Irun-Briz, H. Decker, J. E. Armendariz-Inigo, and J. R. Gonzalez de Mendıvil. Managing transaction conflicts in middleware-based database replication architectures. In SRDS, pages 401–410. IEEE Computer Society, 2006.

[14] J. R. Juarez, J. E. Armendariz, J. R. G. de Mendıvil, F. D. Munoz, and J. R. Garitagoitia. A weak voting database replication protocol providing different isolation levels. In NOTERE'07, 2007.

[15] J. R. Juarez, J. R. Gonzalez de Mendıvil, J. R. Garitagoitia, J. E. Armendariz, and F. D. Munoz. A middleware database replication protocol providing different isolation levels. In EuroMicro-PDP. Work in Progress Session, 2007.

[16] J.R.Juarez-Rodriguez and J.E.Armendariz-Inigo "A Database Replication Protocol Where Multiast Writesets Are Always Commited",The Third International Conference On Availability,Reliability and Security,0-7695-3102-4/08$25.00©2008IEEE DOI 10.1109/ARES.2008.62

[17] J. R. Gonzalez de Mendıvil, J. E. Armendariz, J. R. Garitagoitia, L. Irun, F. D. Munoz, and J. R. Juarez. Nonblocking ROWA protocols implement GSI using SI replicas. Technical Report ITI-ITE-07/10, Instituto Tecnologico de Informatica, 2007.

[18] C. Amza, A. L. Cox, and W. Zwaenepoel. A comparative evaluation of transparent scaling techniques for dynamic content servers. In ICDE, pages 230–241. IEEE-CS, 2005

[19] C. Shen, K. Ramamritham and J.A. Stankovic, "Resource Reclaiming in Multiprocessor Real-Time Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 4, April 1993.

[20] P. A. Bernstein, D.W. Shipman, and J. B. R. Jr. Concurrency control in a system for distributed databases (sdd-1). ACM Trans. Database Syst., 5(1):18–51, 1980.

[21] D. Skeen and D. D. Wright. Increasing availability in partitioned database systems. In PODS 84: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems, pages 290–299, New York, NY, USA, 1984. ACM Press.

Authors

Sanjay Kumar Tiwari received his Master degree in Computer Application in year 2007 presently he is working as Instructor in NSIC, Computer Centre, Ujjarpar, Gorakhpur. He has more than 5 years teaching and professional experience. His area of interest includes DBMS, Discrete Mathematics & Networks. He is pursuing his Ph.D. in Computer Science on "Relative Performance Issue in Distributed Real Time Database For Replicated Data". He has published several papers in National and International conferences and Journals.

Dr. A.K. Sharma received his Master degree in Computer Science in year 1991 and Ph.D. degree in 2005 from IIT, Kharagpur. Presently he is working as Associate Professor in Computer Science and Engineering Department, Madan Mohan Malaviya Engineering College, Gorakhpur. He has more than 23 years teaching experience. His areas of interest include Database Systems, Computer Graphics, and Object Oriented Systems. He has published several papers in National & International conferences & journals.

Vishnu Swaroop received his Master degree in Computer Application in year 2002 presently he is working as Computer Programmer in Computer Science and Engineering Department, Madan Mohan Malaviya Engineering College, Gorakhpur. He has more than 22 years teaching and professional experience. His area of interest includes DBMS, & Networks; he is pursuing his Ph.D. in Computer Science on Data Management in Mobile Distributed Real Time Database. He has published several papers in several National, International conferences and Journals.