Volume 4, Issue 7, July 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Replication Protocol Where Multicast Writesets Never Got Aborted

Ashish Srivastava[1], Udai Shankar[2]
[1]Deptt. Of Computer Application, M.B.S.P.G. College, Gangapur, Varanasi, India
[2]Deptt. Of Computer Science & Engineering M.M.M. university of Technology, Gorakhpur, India

Abstract— Database replication protocols for real time system based on a certification approach are frequently the best ones for achieving good performance. T voting approach achieves a little 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. We propose a new database replication method that shares many qualities 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 along with verification phase incorporated in the new system are not broadcast in our strategy rather than only validation phase. Thus, this new approach certainly reduces the communication traffic and also achieves a good transaction response time (even shorter than those previous strategies associated with only validation phase in some system configurations).

Keywords - Real time system, replicated database, certification protocol, deterministic protocol, weak-voting protocol, validation and verification phase.

I. INTRODUCTION

A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or it is useless. 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 [1] or total order [2] 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 [3], 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 [4]. From those presented there, the two foremost techniques, in performance terms, are: certification-based [5] and weak-voting [6] 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 [7], in order to abort or commit the delivered transaction. In the former case, the writeset is discarded (except for it's 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 [2] 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 behaviour 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) [8] 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, 22], whilst quite a few weak-voting ones .If we consider the previous certification-based protocols one of their key differences is the way in which the application and commitment of an already certified writeset is handled by the given replication protocol. It is important to note that a certified writeset must be committed at all replicas just to ensure the atomicity of a transaction. Hence, some protocols will make use of re-attempt mechanisms whilst others will take advantage of the block detection mechanism already provided by almost every DBMS .

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 [14] 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 replicas 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 [15] 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. Replicas 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 verified 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 [11]. 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 behaviour 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 (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. DETERMINISTIC PROTOCOL

This Section is devoted to explain the Determ-Rep protocol executed by the middleware at a replica Rk (shown in Figure 1). All operations of a transaction Ti are submitted to the middleware (abort operations are ignored for simplicity) corresponding to its delegate replica. The middleware maintains a list (to work) that determines the same scheduling of transactions in the system, i.e. each replica stores the same copy, though not shared, of to work. Here, for the sake of understanding, it is assumed a round robin scheduling based on replica identifiers in Figure 1. In other words, to work is in charge of deciding which replica can send a message or which writeset has to be applied respectively. Initially, it is filled with infinite tuples (just for algorithm presentation purposes) of the form<order, n mod N, n, Φ> with nϵN, sorted by n. All operations are forwarded to the local database replica but the commit operation (step I of Figure 1). Besides, the replica maintains a list (WS list) storing local transactions (Ti.replica = Rk) that have re-quested their commit. Hence, when a transaction requests its commit the writeset is retrieved from the local database replica, if it is an empty one the transaction will be committed, otherwise the transaction (along its writeset) will be stored in WS list. Concurrent to this, the replica must wait for its turn, i.e. the <order, k, n, Φ>, in its own to work (II). If there are no transactions stored in

WS list it will make advance the turn to the next middleware replica, via sending the <next, k, n, Φ> to all replicas. Otherwise, it will multicast (using the reliable service) all the writesets contained in WS list and its content is emptied in a <to_commit, k, n, WS_list>. Upon delivery of these next and to commit messages they are substituted in their proper positions (which is reflected by n) of the to work list of the delivered replica (III). It is important to note that, although it was the first position of the to work contained in the message sender replica, all replicas run at different speed and there could be replicas still handling previous positions of their own to work. Therefore, in the case of reaching the first position a next message it will be deleted so the next position of to work can be executed (IV).

If we take a look on how writesets are applied, there can be several alternatives, though it is an implementation detail, to apply one by one transaction or all of them grouped in a single remote transaction. In Figure 1 it has been followed the former (V). We will distinguish two cases of writeset execution: at its delegate replica or at a remote replica. In the first case the transactions will be directly committed. Whereas in the other case a remote transaction is used to apply and commit the transaction. As it can be inferred, if we are not very careful it will be the case that this transaction can never progress, it may conflict with local transactions and being involved in a local database deadlock. To partially avoid this we stop the execution of write operations in the system (see step I.1.a in Figure 1) when a remote writeset is applied at a replica, i.e. Turning the ws_run variable to true. However, this is not enough to prevent the writeset abortion in the replica; the writeset can be involved in a deadlock with local transactions that already wrote in some data item that intersects with the writeset and be aborted and, hence, it must be re-attempted until its successful completion. This last feature is ensured since a pretty similar block detection mechanism as already presented in [15] can be used to abort all possible conflicting local transactions (VI).

IV. FORECAST 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 1.

Table.1 - Feasibility test of Real Time System

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

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, SEt is the scheduled end time for task Tl 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 Tl ϵ Batch(j) is scheduled on processor Pk during the jth phase and that Tl misses its deadline at execution time. This assumption leads to the following condition:

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

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:

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

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

V. MODIFIED ALGORITHM FOR REPLICATION IN REAL TIME SYSTEM

Initialization:

1. ws_run := false
2. lastcommitted_tid := 0
3. WS_ list: = Φ
4. to_work := {<order, (n mod N), n, Φ >, n ϵ N}

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

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

/* Transactions Abort after the constant of time */

1. 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
2. else if SELECT then
a. execute operation at Rk and return to client
3. else /* COMMIT */
a. Ti.WS := get_ writeset(Tik) from local Rk
b. if Ti.WS = Φ, then commit and return
c. Ti.pre_ commit := true
d. WS_ list := WS_ list <Ti>

Step-II. First <order, k, n, Φ >in to work
1. if WS_ list = Φ then R_ multicast(<next, k, n, Φ>)
2. else R _multicast(<to_ commit, k, n, WS_ list>)

Step-III. Upon receiving m /* first */
/* m is either <next, id, n, Φ > */
/* or <to_ commit, id, n, seq_txns> */
1. Substitute <order, id, n, Φ> by m

Step-IV. First <next, id, n, Φ> in to_ work
1. Remove <next, id, n, Φ> from to work

Step-V. First <to_ commit, id, n, seq_ txns>
in to_ work
1. While seq_txns≠ Φ do
a. T′ := first in seq_txns
b. if T′.replica = k then
- commit T′
- remove <T′> from seq_txns
c. else /* T′ is remote */
- ws_run := true
- apply T′.WS to local Rk
/* T′ may be reattempted */
- commit T′
- remove <T′> from seq_txns
2. Remove <to_commit, id, n, Φ >
from to_work
3. ws_run := false

Step-VI. Upon block between T1 and T2
/* T1.replica ≠ k */
/* T2.replica = k, i.e., local */
1. abort T2
2. if T2.pre_commit = true then
a. remove <T′ > from WS list

Fig.1. Algorithm for Replication in Real Time System

VI. 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 ϵ T, whose delegate replica is Rk with k ϵ N, and 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 1, 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.
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.

Theorem2 (Atomicity of transactions). If a (tocommit,Rn,seq_txns) is processed and committed at a replica Rk with k ∈ 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'ϵN ^ k ≠ k' the replica to analyse. 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.

VII. EXPERIMENTAL RESULTS

We have simulated our deterministic protocol, comparing it with a general certification-based one. To this end, the simulation parameters being considered have been summarized in Table 2. As it can be seen in such table, we have chosen very slow networks for two different configurations: either a LAN or a WAN. 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 seen as a way of implementing a total-order broadcast, as already discussed above. But this leads to a poor-performance way of implementing such a total-order broadcast when the network is slow.

Table.2. Simulation parameters

S.NO.	PARAMETER	VALUES
1.	Replicas	2,4,6,8,10,12,14,16,18 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%

The number of transactions used in each experiment has ensured that the standard deviation is below 3% of the plotted mean values in all figures being presented in this Section. Each transaction consists of an update and a reading sentence . For read-only transactions, the update has been replaced by another reading sentence. 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. Note also the global loads shown in Table 1 refer to the transaction arrival rate. In order to model resource contention we have used two different configurations. In the first one, each machine has 6 open connections for accessing the underlying database. This means that the local DBMS is able to serve 6 concurrent transactions at a time, leading to a contention scenario if the global load exceeds 45N TPS, being N the number of replicas.

.

VIII. RESULTS WITH 6 CONNECTIONS PER NODE

In this series of tests, we have used 6 connections per node, checking the protocols behaviour both in LAN and WAN deployments 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 applications). Besides the transaction completion time in such four cases, we also analyse their abortion rate. Thus it shows the results in the worst case being considered, i.e., when no read-only transactions are included in the simulated load. In the LAN case, 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). Differences are much bigger in the WAN deployment. Although the deterministic protocol is slightly better when only two replicas are considered, its transaction completion time has an increasing trend in all cases when more replicas are added to the system (except with 300 TPS, where adding more replicas has a favourable impact due to its contention reduction).

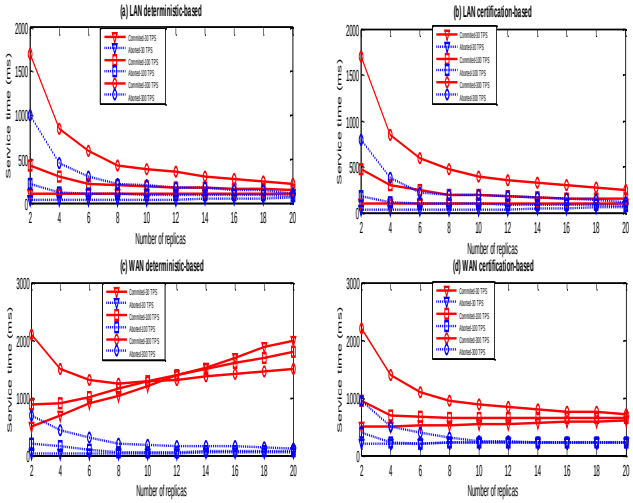

Figure 1Transaction completion time with0% RO-trans

Moreover, the lower is the load; the bigger is the increasing trend. It is worth noting that the deterministic protocol depends a lot on the size of the logical ring. If we assume a system with more than 10 nodes and a very light load, there are few transactions being multicast when the sending privilege arrives to each node, and this leads to a lengthy transaction service time. Indeed, for the lightest load being considered in our simulation (30 TPS), the completion time is almost 4 times bigger when 20 replicas are used. However, the transaction abortion time is shorter in the deterministic protocol than in the certification-based one, in all cases (both when the load and the number of nodes are varied). Note that in the deterministic protocol such transaction abortion times are almost equal in both

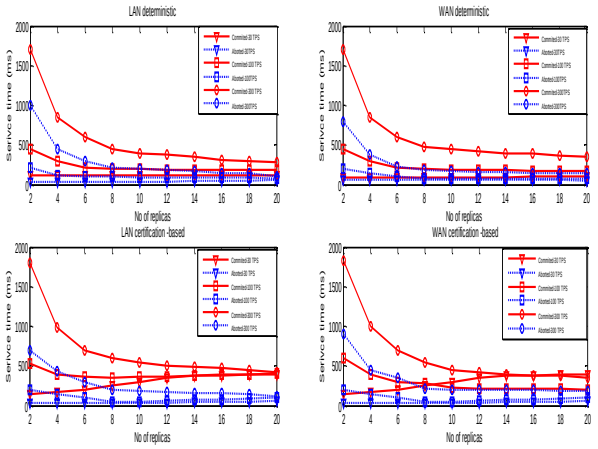

Figure 2- Transaction completion time with80% RO-trans

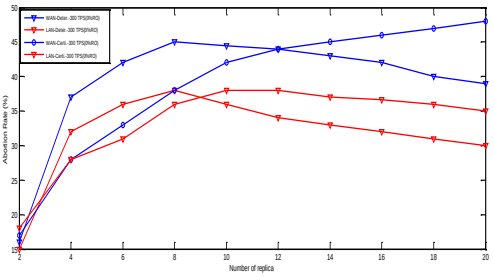

Figure 3- Abortion ratio (Heavily Load) with 0% RO-trans

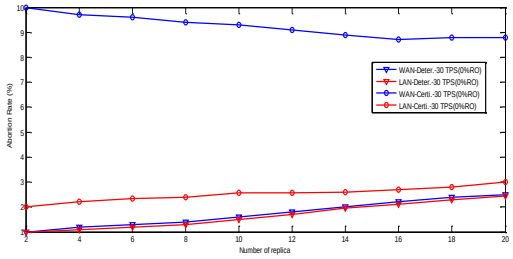

Figure 4-Abortion ratio (Light Load) with 0% RO-trans

kinds of network (the single difference is for a deployment with only two replicas with the heaviest load, where resource contention has caused an increase in the abortion time in the LAN network) due to its very light network traffic. However a certification-based protocol is quite penalized when the network is slow, recall that it requires an atomic broadcast for each completed transaction, and this equally affects the completion time for committed and for aborted transactions. It provides the same set of results for a more realistic load where 80% of transactions are read only. Results for a LAN deployment follow the same trend shown above, i.e., there are no significant differences between both protocols. In the WAN deployment, the differences found when all transactions were read only have been highly reduced in this case. Now, the deterministic protocol provides better or comparable results when less than 7 replicas are

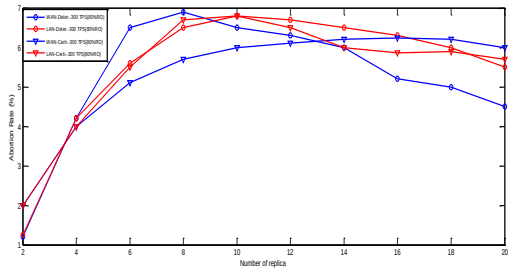

Figure 5-Abortion ratio (Heavily Load) with 80% RO-trans

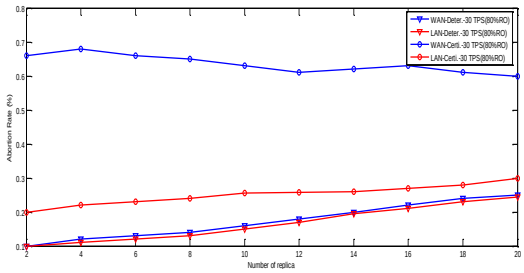

Figure 6-Abortion ratio (Light Load) with 80% RO-trans

considered, whilst in the previous case this only happened with 2 replicas. Moreover, when many replicas have been used, the differences are still big, but lower than in the previous case. Thus, for the worst case (20 replicas and 30 TPS) the mean transaction completion time (for committed transactions) is 413.11 ms in the deterministic protocol and 199.12 ms in the certification-based one; i.e., twice bigger whilst in the previous case it was four times bigger. As expected, the fast abortion time of the deterministic protocol is still maintained with this kind of load. The abortion rates for both kinds of loads are summarized. With a load without read-only transactions, both protocols have a higher abortion rate when a slow network (WAN) is being used. In the deterministic protocol, such differences increase with the load. They are negligible with 30 TPS but exceed a relative 22% with 300 TPS (e.g., with 20 replicas, the respective abortion rates are 35.92% and 29.38%; and thus 35.92/29.38=1.222). On the other hand, in the certification-based protocol the differences between the WAN and LAN results are almost constant with light load (30 TPS), but with medium and heavy load practically disappear when few replicas are used and get progressively increasing when more replicas are considered.

Comparing both protocols, the deterministic one provides the best results for light and medium loads, with bigger differences in the WAN case, and always at least 20% better than the certification-based protocol. However, in the heaviest load case (300 TPS) things are not so clear. The deterministic protocol has its maximum value when 6 replicas are used, both in the LAN and WAN case, whilst the certification-based protocol has its maximum with 20 replicas in the WAN case and with 10 replicas in the LAN one. Due to this, the deterministic protocol is better than the certification-based one when there are more than 10 replicas in the WAN case, and when there are more than 8 in the LAN one. Finally, Figures 4.c and 4.d show 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 load 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 without read-only transactions, but now the curves follow different trends.

IX. FAULT TOLERANCE ISSUES

Right now, we have not covered any issue about the failure of a replica which is something that is more likely to occur in a replicated database system. Moreover, it will be interesting to give a dynamic nature of the composition of replicas in the system (a partially synchronous one). Hence, replicas may fail, re-join or new replicas may come to satisfy some performance needs. The failure and recovery of a replica follows the crash-recovery with partial amnesia failure model [19]. Note that once a transaction has been committed, the underlying DBMS guarantees its persistence, but on-going transactions are lost when a replica fails.

This provides a partial amnesia effect. Management of these issues is handled by the GCS thanks to the Membership Service [20]. This service provides the notion of view, the set of current connected and active nodes. In replicated databases it is important to work under the primary component assumption [20], i.e. a replica is allowed to continue processing transactions provided that there are more than a half replicas connected; otherwise, in general, it is forced to shut down until it becomes part of the primary partition.

The view concept may be considered as a synchronization point for the replicated setting: each time a replica crashes or joins the system a view change event is fired [20] that provides as a report the number of already connected members. Moreover, this event is totally ordered for all replicas that install this new view and it also ensures that replicas contained in the former and the new views deliver the same set of messages; hence, giving the notion of view synchrony. Related to this is the notion of uniform and same view delivery [5] consisting in that if a message is delivered by a replica (faulty or not), it will be eventually delivered to all replicas that install the next view in the former view. All these features let us know which writesets have been applied between failures and joins of nodes and, thus, define what to do in case of a failure or join of a new replica that will be outlined in the following just keeping in mind the protocol.

X. REPLICA FAILURE PROCESS

As it has been said before, the failure of a replica Rj involves firing a view change event. Hence, all nodes will install the new view with the excluded replica. The most straightforward solution is that each alive replica Rk to silently discard the positions of to work associated to Rj. However, one should be more careful about missed writesets by the faulty replica Rj until the view change reporting its failure. However, this is not a very difficult task thanks to the round nature of our protocol. A replica has an auxiliary queue where delivered messages are stored. This queue is pruned each time a new round is started, i.e. it receives a new message that belongs to it. Hence, when a node crashed it is only needed to store the content of this queue. This information will be transferred when it will re-join the system back again. Besides, for the normal scheduling of transactions in the system it is needed to rebuild the to work queue.

XI. THE PROCESS OF RECOVERING A REPLICA

After a replica has crashed, it will eventually re-join the system firing a view change event about this fact. This recovering replica has to apply the (possibly) missed updates on the view it crashed and the updates while it was down. Thanks to the strong virtual synchrony, there is at least one replica that completely contains all the system state. Hence, there is a process to choose a recovered replica among all alive nodes; this is an orthogonal process and we will not discuss here it any further, we will assume that there exists a recovered replica. Upon firing the view change event, like in the previous case, we need to rebuild the to work queue including the recovering replica. The recovered will wait for its turn to send the missed information to the recovering replica. Meanwhile, the recovering will send next messages until its finishes applying the missed updates and keep on discarding messages coming from other available replicas. It is worth noting that the set of missed updates can be inferred quite easily, it is only needed to store the transaction identifier of the last committed transaction before the recovering replica crashed. Thanks to some metadata tables present in some commercial DBMS, such as PostgreSQL, infer the set of registers updated since that transaction and transfer their current state. Concurrently to this, every alive replica will store all writesets delivered that will be compacted [21] in an additional queue called pending WS. Once the recovering is done applying missed updates, it will send a pending message. The delivery of this message to the next replica in to work will send the compacted writesets stored in pending WS to the recovering and, thus, finish the recovery process. As it may be seen, we have followed a two phase recovery process very similar to the one described in the first phase consists in transferring the missed updates while the replica was crashed; and, the second one transfers the missed updates of the current view while the recovery process took place. This last phase serves while establishing a synchronization point with the rest of replicas to consider the recovering replica as alive.

XII. CONCLUSIONS

Our deterministic database replication protocol proposal is able to inherit the best characteristics of both certification based and weak-voting approaches. Thus, like a weak voting protocol, it is able to validate along with verification 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 strategy has been justified. Additionally, its performance has been analysed through simulation, providing a transaction completion time quite similar to that of a certification-based approach (the best one according to previous analysis in some configurations, and with a lower abortion rate). Finally, a recovery strategy for this new kind of replication protocols has also been discussed. It can be easily matched with the regular tasks of this replication proposal.

REFERENCES

[1] 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.

[2] 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.

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

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

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

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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

[15] 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.

[16] 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

[17] 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

[18] J.R. Ju´arez, J.E. Armend´ariz, F.D. Mu˜noz, J.R. Gonz´alez de Mend´ıvil, J.R. Garitagoitia "A Deterministic Database Replication Protocol Where Multicast Writesets Never Got Aborted" Technical Report TR-ITI-ITE-07/15, Instituto Tecnol´ogico de Inform´atica, 46022 Valencia, Spain, June 29, 2007

[19] Flaviu Cristian. Understanding fault-tolerant distributed systems. Commun. ACM, 34(2):56–78, 1991

[20] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. ACM Comput. Surv., 33(4):427–469, 2001.

[21] J. Pla-Civera, M. I. Ruiz-Fuertes, L. H. Garc´ıa-Mu˜noz, and F. D. Mu˜noz-Esco´ı. Optimizing certification-based database recovery. In *6th ISPDC*, Hagenberg, Austria, 2007. IEEE-CS. Acc.for publication.

[22] Sameh Elnikety, Fernando Pedone, and Willy Zwaenopoel. Database replication using generalized snapshot isolation. In *SRDS*. IEEE-CS, 2005.

