Evaluating Certification Protocols in the Partial Database State Machine[*]

A. Correia Jr.[†] A. Sousa J. Pereira R. Oliveira F. Moura

Universidade do Minho

Abstract

Partial replication is an alluring technique to ensure the reliability of very large and geographically distributed databases while, at the same time, offering good performance. Partial replication is done by splitting the database according to application semantics and then by replicating each fragment at a subset of the available sites. Access locality should allow that each transaction needs only a small subset of all sites to execute and commit thus reducing processing and communication overhead associated with replication.

The advantages of partial replication have however to be weighted against the added complexity that is required to manage it. In fact, if the chosen configuration cannot make transactions execute locally or if the overhead of consistency protocols offsets the savings of locality, potential gains cannot be realized. Unfortunately, both these issues are heavily dependent on the application used for benchmarking thus rendering simplistic benchmarks useless.

In this paper we present a detailed analysis of Partial Database State Machine (PDBSM) replication by comparing alternative partial replication protocols with full replication. This is done using a realistic scenario based on a detailed network simulator and access patterns from an industry standard database benchmark. The results obtained allow us to identify the best configuration for typical on-line transaction processing applications.

Keywords: Distributed Databases, Replication, Group Communication, Performance Evaluation.

[*] Research funded by FCT, STRONGREP project (POSI / 41285 / CHS / 2001).

[†]Contact author. email: alfranio@lsd.di.uminho.pt Address: Departamento de Informática -Universidade do Minho, Campus de Gualtar, 4710-057 Braga, PORTUGAL

1 Introduction

Database replication based on group communication has recently been the subject of several research efforts [3, 16, 13, 14, 19]. These have shown that scalability and performance limitations of traditional database replication protocols, mostly involving distributed locking and atomic commitment [10], can be overcome by taking advantage of order and atomicity properties of reliable multicast as offered by group communication [6]. In contrast to the so called lazy or asynchronous replication strategies often implemented in commercial products, this approach preserves 1-copy serializability [4].

One of the solutions that uses this approach is the Database State Machine (DBSM) [16]. Briefly, each transaction request is optimistically executed by a single site. Upon entering the committing stage, the outcome of the transaction is propagated to all replicas using an atomic multicast protocol. A certification procedure is run upon delivery by all sites to determine whether the transaction should be committed or aborted due to conflicts with other concurrently executed transactions. Total order and the determinism of the certification procedure ensure strong consistency. As deterministic execution is confined to the certification procedure, no restrictions impairing performance are imposed on scheduling during the execution stage.

The DBSM is however a full replication protocol: It assumes that the outcome of each transaction is multicasted to all sites, which have complete copies of the database on which the certification procedure can be run. On the other hand, partial replication is done by splitting the database according to application semantics and then by replicating each fragment at a subset of the available sites. During certification, it is expected that sites are only bothered with those fragments that are stored locally. This means however that sites might fail to recognize conflicts related to fragments which are not locally available. Different sites would therefore be able to decide differently and become inconsistent.

This is unfortunate as partial replication is invaluable for the scalability and performance of very large and geographically distributed databases. Fragmentation allows less relevant data items to be replicated by fewer sites. Access locality allows that data items are kept close to those sites that need them more often. If each transaction needs only a small subset of all sites to execute and commit, the processing and communication overhead associated with replication can be reduced.

The DBSM can be extended for partial replication [19] by adding a third stage to the protocol. After certification, each site collects votes from at least a representative from each fragment accessed by the transaction. The transaction is allowed to commit only if no conflicts happen in any of the fragments, restoring strong consistency. The advantages of partial replication have however to be weighted against the added complexity that is required to manage it. In fact, if the chosen configuration cannot make transactions execute locally or if the overhead of consistency protocols offsets the savings of locality, potential gains cannot be realized.

In this paper we address this issue by presenting a detailed analysis of Partial Database State Machine (PDBSM) replication. Specifically, we compare (i) the original DBSM protocol with (ii) a partial replication protocol in which full certification is performed by all sites and (iii) a partial replication protocol with the additional voting phase.

The trade-offs involved are however heavily dependent on the application used for benchmarking, thus rendering simplistic benchmarks useless. Namely, depending on how the database is fragmented and on transaction profiles, the overhead of full certification might compare differently with the voting phase. Our approach is thus to use a realistic scenario based on a detailed network simulator [8] and access patterns from the industry standard TCP-C [22] database benchmark. The results obtained allow us to identify the best configuration for typical on-line transaction processing applications, as well as to precisely characterize the trade-offs involved, thus gathering valuable knowledge on how to fragment the database.

The rest of this paper is structured as follows: Section 2 introduces some important definitions such as the system model, and the distributed data environment. In Section 3 we present an execution model of the Partial Database State Machine along with two alternative termination protocols. In Section 4, we describe the simulation model, the workload pattern used (TPC-C) and analyze the simulation results. In Section 5, we present related works and we conclude in Section 6.

2 System Model and Definitions

We consider a distributed system composed of two completely connected sets $S = \{s_1, ..., s_n\}$ and $C = \{c_1, ..., c_m\}$, respectively of database and client sites. Sites communicate through message passing (i.e., no shared memory). The system is asynchronous in that we make no assumptions about the time it takes for a site to execute a step nor the time it takes for messages to be transmitted.

Sites may only fail by crashing (i.e., no Byzantine failures), and we do not rely on site recovery for correctness. We assume that our asynchronous model is augmented with a failure detector oracle [5] so that Atomic Multicast and View Synchronous Multicast [6] are solvable.

2.1 Databases and Transactions

A relational database $DB = \{R_1, \ldots, R_s\}$ is a set of relations $R_i \subseteq D_1 \times \ldots \times D_q$ defined over data sets not necessarily distinct. Each element $(d_1, d_2, ..., d_q)$ of a relation R_i is called a tuple and each d_i is called an attribute. To uniquely identify each tuple of a relation, we assume the existence of a minimum nonempty set of attributes, called the *primary key*.

The relations of the database can be fragmented horizontally and vertically by means of two operators. The horizontal fragmentation of a relation R_i corresponds to a predicate selection and can be defined as $H(R_i, \sigma) = \{t \mid t \in R_i \wedge \sigma(t)\}$. The vertical fragmentation of R_i is a projection of the relation over a set of attributes and can be defined as $V(R_i, J) = \{t_J | t \in R_i\}$ such that $t_J =< \ldots, d_j, \ldots >_{j \in J}$.

We consider a distributed relational database as a relational database whose relations are distributed among the set S of database sites. This distributed database is given by $DDB \subseteq DB \times S$.

Client sites submit transaction requests to database sites. A transaction represents a sequence of operations of the relational algebra [24, 7], followed by a *commit* or *abort* operation. The result of executing a transaction is a sequence of reads and writes of tuples. The read set of a transaction t, denoted by $RS(t)$, is the set of primary keys identifying the tuples read by t. Its write set, $WS(t)$ is the set of primary keys identifying the tuples written by t, and $WV(t)$, called write values, the set of tuples written by t.

2.2 The Database State Machine

The Database State Machine [17] is based on the deferred update replication technique [4] which reduces the need for distributed coordination among concurrent transactions during their execution. Using this technique, a transaction is locally synchronized during its execution at the database where it initiated according to some concurrency control mechanism [4] (e.g., two-phase locking, multiversion). From a global point of view, the transaction execution is optimistic since there is no coordination with any other database site possibly executing some concurrent transaction. Interaction with other database sites on behalf of the transaction only occurs when the client requests the transaction commit. At this point, a *termination protocol* is started: i) the transaction write values, read and write sets are *atomically propagated* to all database sites, and ii) each database site *certifies* the transactions determining its fate: commit or abort.

The DBSM provides 1-copy-serializability [4] as its consistency criteria [16]. To ensure the same sequence of committed transactions at all database sites the technique requires transactions to be: i) executed only once, and its write values applied to all sites, ii) totally ordered and, iii) deterministically certified and committed.

In order for a database site to certify a committing transaction t, the site must be able to determine which transactions conflict with t. A transaction t' *conflicts with* t if: i) t and t' have conflicting operations and ii) t' does not *precede* t.

Two operations conflict when they are issued by different transactions, access the same data item and at least one of them is a write operation. The precedence relation between transactions t and t' is denoted $t' \rightarrow t$ (i.e., t' precedes t) and defined as: i) if t and t' execute at the same database site, t' precedes t if t' enters the committing state before t; or ii) if t and t' execute at different sites, for example s_i and s_j, respectively, t' precedes t if t' commits at s_i before t enters the committing state at s_i.

3 Partially Replicated Database State Machine

Releasing the assumption that each database site contains a full copy of the database directly impacts both the execution and the certification of transactions. In this section we address the issues raised by partial replication in the Partial Database State Machine (PDBSM). In detail, we address the execution model and two possible termination protocols that deal with partial replication, with either independent or coordinated certification.

3.1 Transaction Execution

From the time it starts until it finishes, a transaction passes through some well-defined states. A transaction is considered to be in the *executing state* as soon as the request is received by an *initiator site* and until a commit operation is issued. The transaction then enters the *committing state* and the distributed termination protocol is started. Unlike the DBSM, the initiator site in a partial replication setting may not be able to locally complete the execution of a transaction. In fact, it is possible that no single site can, if the required fragments are nowhere held together. Therefore, the execution of a transaction t in the PDBSM requires that the initiator site s_i coordinates the distributed processing of t among a set of sites that together contain all the fragments accessed by t. Roughly, the initiator site s_i goes through the following steps [15]: i) it parses each request and rewrites the operations mapping the original database relations into the actual fragments, ii) selects the appropriate sites for each fragment accessed by t, iii) constructs t's execution plan, and iv) starts the distributed execution. Concurrency control is performed locally by each site when executing operations on behalf of the initiator site. In this paper we consider multiversion locking [4].

3.2 Termination Protocol

An issue of major impact for the PDBSM is how the results of the transaction distributed processing are handled. While in the DBSM, the whole set of write values, read and write sets were relevant to all database sites, in a fragmented database this is no longer true. On the contrary, the fragmentation of the database is meant to exploit data and operation locality and therefore the propagation of write values should be restricted to the sites replicating the involved fragments.

With respect to the read and write sets, however, it is not obvious whether they should be propagated to all database sites or just to those containing the relevant fragments. Indeed, this directly influences the certification phase and establishes a trade-off between network usage and protocol latency. If the whole read and write sets of the transaction are fully propagated, then it will enable each site to independently certificate the transaction. Otherwise, if each site is provided with only the parts of the read and write sets regarding the site's fragments, then it can only make a partial judgment and the transaction certification requires a final coordination among all sites. In the following we detail these two termination protocols and in Section 4 we evaluate them.

3.2.1 Independent Certification

With the propagation of the whole read and write sets to all database sites we adopt a termination protocol similar to that of the DBSM, in which each site can independently certify the transactions. As soon as a transaction t enters the committing state, all sites containing database fragments involved in the transaction are requested to *stabilize* the transaction write values. The stabilization of a fragment ensures that all sites are able to participate on the termination protocol and to eventually commit the transaction despite the failure of the sites involved on the transaction execution. By request of the initiator site s_i, each database site s_j involved on the execution reliably multicasts the transaction write values of the fragments it is responsible for to all sites replicating these fragments. The initiator s_i receives a stabilization acknowledgment, in the form of read and write sets, from s_j. Should s_j fail, s_i may unilaterally decide to abort the transaction. Notice that view synchrony ensures that there is no ambiguity on which transactions should be aborted due to site failures during stabilization.

Once the initiator site gathers all the acknowledgments, it atomically multicasts the transaction read and write sets to all database sites. This message totally orders the certification of t, and thus upon delivery of the message, along with the read and write sets of previously certified transactions, each site has the necessary knowledge to certify t. If t passes the certification test, all write values of t previously obtained during t's stabilization phase are applied to the database and t passes to the *committed state*. Otherwise, t passes to the *aborted state*. Transactions in the executing state holding locks on data items updated by t are aborted when t commits.

The cost of independent certification is given by the cost in network bandwidth of propagating the whole read and write sets to all sites, plus the cost, at each site, of keeping these read and write sets while required for the certification of pending transactions, and finally the cost of certifying the whole transaction at each site. From these, our main concern should actually be on the network usage. The read and write sets of a transaction t can be discarded as soon as t is known to precede every pending transaction, that is, any transaction in the executing or committing state. The difference in the cost of doing total or partial certification is almost negligible.

3.2.2 Coordinated Certification

On the other hand, to fully exploit data locality we restrict the propagation of the transaction read and write sets to the database sites replicating the corresponding fragments. The knowledge required to certify a transaction becomes itself fragmented and each site may only be able to certify part of the transaction. Therefore, a final coordination protocol is required.

Once the transaction reaches the committing state the initiator site requests all sites containing database fragments involved in the transaction execution to stabilize. Each one of these sites reliably multicasts the read and write sets and write values of the transaction's fragments it is responsible for to all sites replicating these fragments and acknowledges the end of the stabilization.

Once the initiator site gathers all the acknowledgments it atomically multicasts a message to all sites to totally order the certification of t. Upon the delivery of the message, each database site s_j certifies t against the fragments it replicates and votes on a Resilient Atomic Commitment (RAC) protocol to decide the final state of t. This protocol allows participants to decide *commit* even if some of the replicas of a fragment read or written by the transaction are suspected to have failed, as a single representative from each fragment suffices. Resilient Atomic Commit satisfies the same agreement and termination properties of Weak Non-Blocking Atomic Commit [11] and is defined as follows:

Agreement: No two participants decide differently.

Termination: Every correct participant eventually decides.

Validity: If a site decides *commit* for t, then for each fragment accessed by t there is at least a site s_i replicating it that voted *yes* for t.

Non-triviality: If for each fragment accessed by t there is at least a site s_i replicating it that votes *yes* for t and is not suspected, then every correct site eventually decides *commit* for t.

If the outcome of the RAC is commit, then all write values of t previously obtained during t stabilization phase are applied to the database and t passes to the *committed state*. Otherwise, t passes to the *aborted state*. If t commits, at each database site, transactions in the executing state holding locks on data items updated by t are aborted.

The RAC protocol is trivially implemented by having each site multicast its vote. A site decides upon receiving a vote from at least a representative of each database fragment. Evaluating the cost of a protocol by its latency degree, as introduced in [18], we easily conclude that this is the implementation of RAC offering the lowest cost.

3.2.3 Implementation Issues

In this section we point out several optimizations to the PDBSM termination protocols. We chose to present them separately to avoid cluttering the description of the protocols with performance oriented concerns. All of these optimizations were included in our PDBSM prototype and contribute to the experience results presented in Section 4.

In order to allow independent certification and ensure 1-copy-serializability every database site must have access to both read and write sets [16]. However, sometimes it becomes prohibitive to send the read set due to its large size. This issue can be circumvented by the definition of a per relation threshold, above which it is assumed that the read set corresponds to the entire relation. This solution can enormously reduce network bandwidth consumption and transaction latency, at the cost of possibly increasing the number of transaction aborts due to the coarser grain in conflict detection.

An optimization that has been used in [19] and can also be applied in large-scale settings [20] is the use of the Fast Atomic Broadcast protocol. The idea behind this protocol is simple and consists in the early delivery of transactions with a tentative order, allowing an optimistic certification to run concurrently to the total ordering protocol. Whenever the tentative order matches the final delivery order, this allows to overlap the final delivery of the transaction with its certification.

A structural optimization that can be applied to the coordinated certification protocol (Section 3.2.2) consists in eliminating the stabilization phase prior to the multicast totally ordering the transaction. Avoiding the stabilization acknowledgment messages across a long distance link can definitely reduce the latency of the protocol. To do so, the multicast totally ordering the transaction is done right after the commit request from the client and the write values stabilization postponed. When a site s_j responsible for executing the transaction on fragment f delivers the totally ordered message, s_j certifies the transaction against f and determines its vote. Its vote should be the same for all the database sites containing copies of f. Therefore, instead of stabilizing t's write values, and read and write sets among the copies of f, it uses view synchronous multicast to stabilize the write values and the fragment's vote for t. Upon the stabilization, each site containing f skips the certification phase and can immediately vote on the RAC protocol. The trade-off here consists in delaying the point where the termination protocol becomes resilient to the crash of s_j, from a stabilization phase prior to the atomic multicast totally ordering t to the beginning of the RAC.

Another optimization regarding the stabilization phase can be applied to fully replicated fragments. The transaction write values regarding these fragments can be included in the atomic multicast message which already ensures data stability upon delivery. This avoids the preliminary stabilization step.

4 Experimental Results

In this section, we use a simulation model to compare a fully replicated DBSM with partial replication using the PDBSM. This is done with both independent and coordinated certification. We start by briefly describing the simulation model and the traffic used. Afterwards we present and discuss the results.

4.1 Simulation Model

The model used combines simulated components for the network, database execution and traffic generation with real implementation of group communication and certification protocols. This provides a centralized environment for testing and evaluating the performance of those components in which we are interested, without the burden of a complete implementation and experimental setup. It allows also to compare different configurations using the exact same input load replayed from trace files. A detailed analysis of this simulation model is beyond the scope of this paper and is presented elsewhere [21]. Nonetheless, we briefly describe the operation of the model for completeness.

Traffic is injected by a set of clients. Each client is single threaded and communicates with only one server. A trace is used to drive the client, specifying the sequence of transactions to be submitted. The trace determines also the amount of time to wait before each transaction is submitted, thus modeling think-time. Each transaction is specified as a sequence of operations that consume resources. Namely, the request specifies CPU time, data to be read or written from disk, and the identifiers of data items

Relations	**Cardinality**	**Tuple Length**	**Relation Size**
Warehouse	1	89 bytes	0.089 *k* bytes
District	10	95 bytes	0.950 *k* bytes
Customer	30 *k*	655 bytes	19650 *k* bytes
History	30 *k*	46 bytes	1380 *k* bytes
Order	30 *k*	24 bytes	720 *k* bytes
New Order	9 *k*	8 bytes	72 *k* bytes
Order Line	300 *k*	54 bytes	16200 *k* bytes
Stock	100 *k*	306 bytes	30600 *k* bytes
Item	100 *k*	82 bytes	8200 *k* bytes

Table 1: TPC-C Relations (*k* is 1000)

to be used for concurrency control. These values are obtained by profiling a real database server when submitted to the desired load. The database server is therefore modeled as a corresponding collection of resources, operating under the direction of a centralized scheduler. Upon reception of a transaction, operations are sequentially scheduled to execute on corresponding resources. After the transaction commits or aborts, a reply is issued back to the originating client.

When a transaction enters the committing state, the associated information is handed to the certification protocol described in the previous section and is implemented as real code. The time consumed by real code is accounted for and used to update the simulation time and usage of CPU resources, thus realistically combining simulated and real components [2]. The underlying network model is also simulated, allowing testing using arbitrary network topologies [8].

4.2 Application Profile

The application profile used is based on TPC-C [22], the industry standard on-line transaction processing (OLTP) benchmark. TPC-C mimics a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. The traffic is a mixture of read-only and update intensive transactions. The database relations and associated information are presented in Table 1 along with initial sizes of relations for 10 clients. Notice that, according to TPC-C, an additional warehouse should be configured for each additional 10 clients. The initial sizes of tables are also dependent on the number of configured clients.

This application scenario can easily be extended to consider partial replication. Specifically, we consider horizontal fragmentation of tables according to the warehouse. The rationale for this is that

Relations	New Order	Payment	Order Status	Deliver	Stock Level
wharehouse	1 /	1 / 1	/	/	/
district	1 / 1	1 / 1	/	/	1 /
customer	1 /	4 / 2	3 /	14 / 14	/
order	/ 1	/	1 /	14 / 14	/
new order	/ 1	/	/	10998 / 10	/
order line	/ 9	/	10 /	149 / 149	9 /
stock	9 / 9	/	/	/	9 /
item	9 /	/	/	/	/
history	/	/ 1	/	/	/

Table 2: Average number of tuples read/written by each transaction type.

these need to be replicated only locally within the warehouse itself and not globally. The exceptions are *Warehouse*, *Stock* and *Item* which are globally replicated to preserve original application semantics in which an order may be serviced by stock from any warehouse.

An emulated client can request five different transactions types as follows: *New Order*, adding a new order into the system; and *Payment*, updating the customer's balance, district and warehouse statistics. Both these transactions occurring with a probability of 44%. With a probability of 4% each, there are: *Order Status*, returning a given customer latest order; *Delivery*, recording the delivery of products; and *Stock Level*, determining the number of recently sold items that have a stock level below a specified threshold. Table 2 presents the average number of tuples read or written from each of the relations accessed, in a run with 20 emulated clients.

Notice that TPC-C is used by us only as the basis for a realistic application scenario for evaluating PDBSM design decisions and not as a benchmark. The constraints mandated for throughput, performance, wait time, response time, screen load and background execution of transactions are not considered here and thus the results are not comparable with results obtained with TPC-C.

4.3 Analysis

We outline in this section an analysis of the bandwidth consumption for the termination protocols presented. To compare the protocols we consider a network setting that privileges access locality. The network is composed of a wide area network (WAN) with low bandwidth and high latency, aggregating several local area networks (LANs) with much higher bandwidth and much lower latency. We assume that all the replicas of a fragment are in a LAN and thus admit that the bandwidth requirements for data propagation between copies of the same fragment are irrelevant when compared with the effect of traffic crossing long distance links.

The transactions read set, write set, and write values have been divided in two subsets: (i) a subset of fully replicated data items called RS_G, WS_G and WV_G, and (ii) a subset containing partially replicated items called RS_L, WS_L and WV_L. We represent by RAC the bandwidth required by the RAC protocol.

The following formulas present the required bandwidth for the termination protocols proposed:

$$\text{DBSM} \equiv RS_G + WS_G + WV_G + RS_L + WS_L + WV_L \tag{1}$$

$$\text{PDBSM} \equiv RS_G + WS_G + WV_G + RS_L + WS_L \tag{2}$$

$$\text{PDBSM + RAC} \equiv RS_G + WS_G + WV_G + RAC \tag{3}$$

Comparing formulas 1 and 2, it can be easily seen that, in the proposed network setting, the PDBSM protocol using independent certification has lower bandwidth consumption as write values never leave LANs and thus never cross long distance links.

Similarly, from formulas 1 and 3, PDBSM protocol using coordinated certification is expected to outperform the DBSM protocol as long as the RAC's required bandwidth does not exceed the requirements for propagating the read and write sets, and the write values of partially replicated fragments.

Finally, formulas 2 and 3, reveal that coordinated certification is preferable when the bandwidth required for the RAC does not exceed that for transmitting the read and write sets of partially replicated fragments. Specifically, our simulation model has 6 database sites, 3 in each side of the long distance link, and we use multisend to broadcast messages. Therefore, every atomic broadcast implies that 3 times the estimated atomic broadcast message size crosses the long distance link (we ignore the message from the coordinator establishing the message order as it will be the same in all protocols).

Considering the workload used in the simulation and characterized in Tables 1 and 2, the expected size of the atomic broadcast messages for each transaction when using DBSM, partial replication with independent certification (PDBSM) or partial replication with coordinated certification (PDBSM with

		RS	WS	WV	Total
DBSM	New Order	168	168	3367	3703
	Payment	48	40	1540	1628
	Order Status	112	0	0	112
	Delivery	1424	1496	17632	20552
	Stock Level	152	0	0	152
PDBSM	New Order	168	168	2754	3090
	Payment	48	40	89	177
	Order Status	112	0	0	112
	Delivery	1424	1496	0	2920
	Stock Level	152	0	0	152
PDBSM with RAC	New Order	152	72	2754	2978
	Payment	8	8	89	105
	Order Status	0	0	0	0
	Delivery	0	0	0	0
	Stock Level	72	0	0	72

Table 3: Expected message size.

	DBSM	PDBSM	PDBSM with RAC
Size	9534.8	4694.5	$4078.2 + 18 \times RAC$

Table 4: Average transaction required size.

RAC) are presented in Table 3. This values are obtained using equations 1, 2 and 3, respectively. A threshold of 150 tuples is used for the read set.

Using this information along with the frequency probability of each transaction we estimate the per transaction average size for each of the protocols. Considering that the traces used as simulation input generates, on average, 3 new transactions per second, we can estimate the minimum bandwidth for data transmission required for each protocol. The values of transaction size are presented in Table 4 and exclude: IP, group management, protocol management and failure detection overheads.

The values in Table 4 suggest that partial replication should present better results than full replication. It should require about half the bandwidth of a fully replicated scenario. In the PDBSM with RAC protocol this analysis depends on the message overheads since it requires 18 messages per RAC, which means that just in IP headers, which are, about 40 bytes per message it requires 720 bytes. The values presented led us to conclude that DBSM consumes more network resources than PDBSM and that PDBSM should perform better when there is contention in the network.

Until now we have analyzed the required bandwidths but have said nothing about the expected latencies of the protocols we are evaluating. Every protocol starts broadcasting the transaction using the fast

	DBSM		PDBSM		PDBSM with RAC	
Link (Kbps)	256	512	256	512	256	512
Latency (ms)	539.5	164.5	286.2	164.2	1572	223
Abort Rate (%)	21	8	13	8	34	13

Table 5: Simulation Results.

atomic broadcast protocol, and as this protocol propagates the messages concurrently with the ordering mechanism, we expect that it will mask the differences in latency that should happen due to message size. In a network without network congestion we expect that atomic broadcast contribution for the final latency will be the same in all protocols. In such a scenario, PDBSM should be marginally better than DBSM. The PDBSM with RAC protocol on the other hand incurs in the additional latency of the RAC protocol and should present higher latencies.

4.4 Results

Simulation results have been obtained with a wide area network model, composed by a backbone network with several routers connected through long distance links to which the 2 leaf networks (100 Mbps LANs), containing each a group of 3 simulated database sites are connected. The network latency between two hosts in different leafs is 60 ms. Each site is configured according to the profiling of a server equipped with 2 Pentium III 1GHz processors, 1GB RAM and a storage device with maximum throughput of 9.486MBps[1]. The lock behavior is based in PostgreSQL's multiversion and a cache hit ratio of 1 is assumed.

In order to evaluate our expectations about transaction latencies we ran a set of simulations using long distance links of 256 and 512 Kbps, whose latency and abort rates are registered in Table 5. In the simulation with bandwidth of 512 Kbps, DBSM and PDBSM present similar results. This confirms our expectation that the additional amount of data required by the DBSM protocol should be masked by the use of fast atomic broadcast. Also as expected PDBSM with RAC has higher latencies due to the overhead of the RAC protocol. The higher abort rate of PDBSM with RAC is also a consequence of its higher latency which increases the number of concurrent transactions in the system and consequently the probability of conflicts.

[1]This value has been measured during a run of a TPC-C benchmark syncing after each finished transaction.

When the bandwidth is reduced to 256 Kbps, PDBSM is clearly the best choice. In this case there is network contention and as such the atomic broadcast protocol can not mask the extra bandwidth required by the DBSM protocol, which presents a latency almost 89% larger than PDBSM and an abort rate about 60% larger. The PDBSM with RAC results are worse than DBSM due to the higher number of messages and by network congestion prevention mechanisms, which drop messages requiring its retransmission and as such increasing the latencies and consequently the abort rates.

5 Related Work

Most of previous work on database replication using group communication [3, 16, 13, 14] concentrates on *full replication* strategies. Along with the assumption of the deterministic processing of transactions at every replica, the resulting protocols, characterized as *non voting* [23], take advantage of not requiring a final agreement protocol.

To the best of our knowledge, the works in [1, 9, 19, 12] are the only ones to consider partial database replication. Alonso in [1] discusses future trends for partial database replication based on atomic broadcast, stating that solutions for full replicated scenarios may not be solutions for partially replicated ones. The work in [9] considers an object-oriented database and uses group communication primitives to immediately broadcast read operations to all replicas of an item, and broadcast all write operations along with the transaction commit request. Transaction atomicity is ensured by a final atomic commit protocol. By contrast, we eliminate replica interaction during transaction processing in both approaches presented. In [19], it is assumed that any transaction can be entirely executed in any database site thus not considering the distributed execution of transactions. Moreover, in [19] the termination protocol fully propagates read and write sets and yet uses a final agreement protocol.

The work of [12] uses epidemic protocols for implementing a dynamic, adaptive replication schema. Transaction execution is entirely local by temporarily caching relations not replicated at the initiator's site. Similarly to the PDBSM, transactions are executed optimistically without any coordination between database sites. Unlike the PDBSM that uses atomic multicast to tottaly order the certification of transactions, in [12], the transaction read and write sets are epidemically propagated and the transaction "certifies" at each site. By contrast to the PDBSM, a conflict between two transactions dictates the abortion of both.

Except for [21], all the analysis of previous work uses simple workloads which do not realistically reproduce the behavior of concurrency control on performance or of certification on the number of aborted transactions. The accurate simulation of bandwidth usage in the network is also dependent on the realism of the model.

Other important aspect of our work, is the simulation model. Usually, the researches mentioned here implement full simulation, except [14] that uses a real DBSM implementation. However, a real implementation makes difficult the setup and management of different experiments (e.g., exploit WAN environments). The combination of simulated and real components [2] give us the possibility to use different environment scenarios but focus on the components under study.

6 Conclusion

We analyze and evaluate two alternative protocols to extend the Database State Machine (DBSM) [16] for partial replication. Our main goal is to reduce communication overhead associated with the full replication by exploiting application data distribution and access locality.

In detail, the first alternative proposed propagates the read set and write set of each transaction executed to all replicas, while ensuring that the propagation of the updates is restrited to interested sites. Our experimental results show that partial replication reduces the used bandwidth. The second protocol ensures that both read and write sets as well as updates are all restricted to interested sites. Our experimental results show that this approach is still better than full replication. The overhead of the required final coordination protocol of this approach makes increases the required bandwidth. These results were obtained with realistic traffic patterned after the industry standard TPC-C benchmark in which some frequently used tables are fully replicated to preserve application semantics. Analysis shows that if a more thoroughly fragmented database with strong access locality would make the second approach preferable.

The overall evaluation of PDBSM replication using a detailed model of an wide area network presented in this paper, is however sufficient to show that partial replication using this approach is useful for large and geographically distributed databases.

Acknowledgments

We thank Luciano Rocha and Luis Soares for their helpful comments and suggestions on the elaboration of this paper.

References

[1] G. Alonso. Partial database replication and group communication primitives (extended abstract). In *Proceedings of the* 2^{nd} *European Research Seminar on Advances in Distributed Systems (ERSADS'97)*, 1997.

[2] G. Alvarez and F. Cristian. Applying simulation to the design and performance evaluation of fault-tolerant systems. In *IEEE Intl. Symp. Reliable Distributed Systems*, 1997.

[3] Y. Amir, D. Dolev, P. Melliar-Smith, and L. Moser. Robust and efficient replication using group communication. Technical Report CS94-20, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, 1994.

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

[5] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2), 1996.

[6] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4), December 2001.

[7] T. Connolly, C. Begg, and A. Strachan. *Database Systems: A Pratical Approach to Design, Implementation and Management*. Addison-Wesley, 1998.

[8] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski. Towards realistic million-node internet simulation. In *Intl. Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1999.

[9] U. Fritzke and P. Ingels. Systéme transactionnel pour donnés partiellement dupliqués, fondé sur la communication de groupes. Technical Report 1322, INRISA, Rennes, France, 2000.

[10] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of theACM SIGMOD International Conference on Management of Data*, volume 25, 2 of *ACM SIGMOD Record*. ACM Press, June 1996.

[11] R. Guerraoui. Revisiting the relationship between non-blocking atomic commitment and consensus. In *Proceedings of the 9th Intl. Workshop on Distributed Algorithms (WDAG-9)*, LNCS 972. Springer-Verlag, 1995.

[12] J.-A. Holliday, D. Agrawal, and A. El Abbadi. Partial database replication using epidemic communication. In *IEEE Intl. Conf. Distributed Computing Systems*. IEEE, 2002.

[13] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *IEEE Intl. Conf. Distributed Computing Systems*, 1998.

[14] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In *Proceedings of 26th Intl. Conf. Very Large Data Bases (VLDB 2000)*. Morgan Kaufmann, 2000.

[15] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4), 2000.

[16] F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, Département d'Informatique, École Polytechnique Fédérale de Lausanne, 1999.

[17] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. Technical Report SSC/1999/008, École Polytechnique Fédérale de Lausanne, Switzerland, 1999.

[18] A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3), 1997.

[19] A. Sousa, F. Pedone, R. Oliveira, and F. Moura. Partial replication in the database state machine. In *IEEE Intl. Symp. Network Computing and Applications*. IEEE Computer Science, 2001.

[20] A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic total order in wide area networks. In *Proc. 21st IEEE Symposium on Reliable Distributed Systems*, pages 190–199. IEEE CS, October 2002.

[21] A. Sousa, J. Pereira, L. Soares, A. Correia Jr, L. Rocha, R. Oliveira, and F. Moura. Evaluating the performance of the database state machine (DBSM). Technical report, Universidade do Minho, Departamento de Informática, 2003.

[22] Transaction Processing Performance Council (TPC). TPC benchmark C standard specification revision 5.0, 2001.

[23] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Database replication techniques: a three parameter classification. In *Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000)*, pages 206–215, Nürnberg, Germany, October 2000. IEEE Computer Society.

[24] M. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall International, 1999.