Evaluating Certification Protocols in the Partial Database State Machine*

A. Sousa, A. Correia Jr., F. Moura, J. Pereira, R. Oliveira

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. By correctly exploiting access locality most transactions become confined to a small subset of the database replicas thus reducing processing, storage access 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 replica configuration prevents the local execution of transactions or if the overhead of consistency protocols offsets the savings of locality, potential gains cannot be realized. These issues are heavily dependent on the application used for evaluation and render 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.

1 Introduction

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

One solution using this approach is the Database State Machine (DBSM) [**?**]. Briefly, each transaction request is optimistically executed by a single replica. 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 replicas 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 a full replication protocol: It assumes that the outcome of each transaction is multicast to all replicas, which have entire 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 the application semantics and then by replicating each fragment at a subset of the available replicas. During certification, it is expected that replicas are only bothered with those fragments that are stored locally. This means however that replicas might fail to recognize conflicts related to fragments which are not locally available. Different replicas would therefore be able to decide differently and become inconsistent.

This is unfortunate, since 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 replicas. Access locality allows data items to be kept close to those replicas that need them more often. Therefore, if each transaction needs only a small subset of all replicas to execute and commit, the processing, storage access and communication overhead associated with replication can be reduced.

The DBSM can be extended for partial replication [**?**] by adding a third stage to the protocol. After certification, each replica collects votes from at least a representative from each fragment accessed by the transaction. The transac-

* Research funded by EU, GORDA project (FP6-IST/004758).

tion is allowed to commit only if no conflicts happen in any of the fragments, guaranteeing 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 does not allow transactions to 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 the Partial Database State Machine (PDBSM) replication. For this, we compare the original DBSM protocol with (*i*) a partial replication protocol in which full certification is performed by all replicas and (*ii*) a partial replication protocol with the additional voting phase.

The trade-offs involved are however heavily dependent on the application used for evaluation, 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 [**?**] and access patterns from the industry standard TCP-C [**?**] 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 tradeoffs 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 work 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 replicas and clients. They communicate through message passing. The system is asynchronous in that we make no assumptions about the time it takes for a replica to execute a step nor the time it takes for messages to be transmitted.

Replicas may only fail by crashing and we do not rely on replica recovery for correctness. We assume that our asynchronous model is augmented with a failure detector oracle [**?**] so that Atomic Multicast and View Synchronous Multicast [**?**] are implementable.

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 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 replicas. This distributed database is given by $DDB \subseteq DB \times S$.

Clients submit transaction requests to database replicas. A transaction represents a sequence of operations of the relational algebra [**?**, **?**], 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 [**?**] is based on the deferred update replication technique [**?**] which reduces the need for distributed coordination among concurrent transactions during their execution. Using this technique, a transaction is locally synchronised during its execution at the database where it initiated according to some concurrency control mechanism [**?**] (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 replica possibly executing some concurrent transaction. Interaction with other database replicas 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 replicas, and ii) each database replica *certifies* the transactions determining its fate: commit or abort.

The DBSM provides 1-copy-serializability [**?**] as its consistency criteria. To ensure the same sequence of committed transactions at all database replicas the technique requires transactions to be: i) executed only once, and its write val-

ues applied to all replicas, ii) totally ordered and, iii) deterministically certified and committed.

In order for a database replica to certify a committing transaction t, the replica 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 replica, t' precedes t if t' enters the committing state before t; or ii) if t and t' execute at different replicas, 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 replica 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 the *executing replica* and until a commit operation is issued. The transaction then enters the *committing state* and the distributed termination protocol is started. Like in DBSM, we consider that the replica executing the transaction is able to locally complete the execution of a transaction.

In a PDBSM scenario we should consider the distributed processing of t among a set of replicas that together contain all the fragments accessed by t. We have chosen to not include this possibility here as, in a well fragmented database this should happen rarely, and we do not intend to measure the impact of distributed execution on transaction latency.

3.2 Termination Protocol

An issue of major impact for the PDBSM is how the results of the transaction processing are handled. While in the DBSM, the whole set of write values, read and write sets are relevant to all database replicas, 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 replicas replicating the involved fragments. This is done by the executing replica when entering the committing phase.

With respect to the read and write sets, however, it is not obvious whether they should be propagated to all database replicas 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 replica to independently certificate the transaction. Otherwise, if each replica is provided with only the parts of the read and write sets regarding the replica's fragments, then it can only make a partial judgement and the transaction certification requires a final coordination among all replicas (i.e, voting phase). 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 replicas, we adopt a termination protocol similar to that of the DBSM, in which each replica can independently certify the transactions.

On entering the committing phase the executing replica, after sending the write values to the replicas replicating them, atomically multicast the transaction's read and write sets to all database replicas. This message totally orders the certification of t. Upon delivery of this message, along with the write set of previously certified transactions, each replica has the necessary knowledge to certify t. If t passes the certification test, all write values of t are applied to the database and t passes to the *committed state*. Otherwise, t passes to the *aborted state*. Transactions in the executing sate 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 replicas, plus the cost of keeping this write set while required for the certification of pending transactions, and finally the cost of certifying the whole transaction at each replica. From these, the main concern is actually on the network usage. The write set 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 replicas replicating the corresponding fragments. The knowledge required to certify a transaction becomes itself fragmented and each replica may only be able to certify part of the transaction. Therefore, a final coordination protocol is required.

Once again, after sending the write values, the executing replica atomically multicasts a message to all replicas to totally order the certification of t. Upon the delivery of this message, each database replica 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. Resilient Atomic Commit satisfies the same agreement and termination properties of Weak Non-Blocking Atomic Commit [**?**] and is defined as follows:

- **Agreement:** No two participants decide differently.
- **Termination:** Every correct participant eventually decides.
- **Validity:** If a replica decides *commit* for t, then for each fragment accessed by t there is at least a replica s_i replicating it that voted *yes* for t.
- **Non-triviality:** If for each fragment accessed by t there is at least a replica s_i replicating it that votes *yes* for t and is not suspected, then every correct replica eventually decides *commit* for t.

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

Under the assumption that each fragment is replicated by a replica that does not fail, the RAC protocol is trivially implemented by having each replica multicasting its vote [**?**]. A replica decides upon receiving a vote from at least a representative of each database fragment.

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 prototypes and contribute to the experience results presented in Section 4.

In order to allow independent certification and ensure 1-copy-serializability every database replica must have access to both read and write sets [**?**]. 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.

4 Experimental Results

In this section, we use a simulation model to compare a fully replicated DBSM with a partially replicated one using the PDBSM. We start by briefly describing the simulation model and the traffic used. Afterwards we present and discuss the results.

4.1 Simulation Model

To evaluate the protocols we use a hybrid simulation environment that combines simulated and real components [**?**]. The key components, the replication and the group communication protocols, are real implementations while both the database engine and the network are simulated.

In detail, we use a centralized simulation runtime based on the standard Scalable Simulation Framework (SSF) [**?**], which provides a simple yet effective infrastructure for discrete-event simulation. Simulation models are built as libraries that can be reused. This is the case of the SSFNet [**?**] framework, which models network components (e.g. network interface cards and links), operating system components (e.g. protocol stacks), and applications (e.g. traffic analyzers). Complex network models can be configured using these components, mimicking existing networks or exploring particularly large or interesting topologies.

To combine the simulated components with the real implementations the execution of the real software components is timed with a profiling timer [**?**] and the result is used to mark the simulated CPU busy during the corresponding period, thus preventing other jobs, real or simulated, to be attributed simultaneously to the same CPU. The simulated components are configured according to the equipment and scenarios chosen for testing.

The database server handles multiple clients and is modeled as a scheduler and a collection of resources, such as storage and CPUs, and a concurrency control module.

Each transaction is modeled as a sequence of operations: *i*) fetch a data item; *ii*) do some processing; *iii*) write back a

data item. Upon receiving a transaction request each operation is scheduled to execute on the corresponding resource. The processing time of each operation is previously obtained by profiling a real database server.

A database client is attached to a database server and produces a stream of transaction requests. After each request is issued, the client blocks until the server replies, thus modeling a single threaded client process. After receiving a reply, the client is then paused for some amount of time (thinking time) before issuing the next transaction request.

To determine the read-set and write-set of a transaction's execution, the database is modeled as a set of histograms [**?**]. The transactions' statements are executed against this model and the read-set, write-set and write-values are extracted to build the transaction model that is injected into the database server. In our case, this modeling is rather straightforward as the database is very well defined by the TPC-C [**?**] workload that we use for all tests. Moreover, as all the transactions specified by TPC-C can be reduced to SPJ queries, the read-set extraction is quite simple.

4.2 Application Profile

Each database request is generated according to the TPC-C benchmark [**?**]. TPC-C is the industry standard online transaction processing benchmark. It mimics a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. TPC-C specifies a precise set of relations (Warehouse, District, Customer, Item, Stock, Orders, OrderLine, NewOrder and History) and the size of the database as a function of the number of desired clients. The benchmark determines 10 clients per warehouse and, as an example, for 2000 clients, the database contains around 10^9 tuples, each ranging from 8 to 655 bytes. The traffic is a mixture of read-only and update intensive transactions. A client can request transactions of five different types: *NewOrder*, adds a new order into the system (with 44% of the occurrences); *Payment*, updates the customer's balance, district and warehouse statistics (44%); *OrderStatus*, returns a given customer latest order (4%); *Delivery*, records the delivery of products (4%); *StockLevel*, determines the number of recently sold items that have a stock level below a specified threshold (4%). The *NewOrder*, *Payment* and *Delivery* are update transactions while the others are read-only.

The database model has been configured using the transactions' processing time of a profiled version of PostgreSQL 7.4.6 under the TPC-C workload. From the TPC-C benchmark we only use the specified workload, the constraints on throughput, performance, screen load and background execution of transactions are not taken into account.

4.3 Resource Analysis

We outline in this section an analysis of resource consumption, namely, bandwidth, storage and processor, for the termination protocols presented. The network is composed of a wide area network (WAN) with moderate bandwidth and high latency, aggregating several (n) local area networks (LANs), each with several (m) replicas, 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 amount of data exchanged among the replicas during the execution of the RAC protocol.

The following formulas present the amount of data, crossing the LAN boundaries, that has to be transferred among replicas while executing the termination protocols proposed:

$$\text{DBSM} \equiv \sum_{i=1}^{m.(n-1)} (RS_{G_i} + WS_{G_i}) + \sum_{i=1}^{m.(n-1)} (RS_{L_i} + WS_{L_i}) + \sum_{i=1}^{m.(n-1)} (WV_{G_i}) + \sum_{i=1}^{m.(n-1)} (WV_{L_i}) \quad (1)$$

$$\text{PDBSM} \equiv \sum_{i=1}^{m.(n-1)} (RS_{G_i} + WS_{G_i}) + \sum_{i=1}^{m.(n-1)} (RS_{L_i} + WS_{L_i}) + \sum_{i=1}^{m.(n-1)} (WV_{G_i}) \quad (2)$$

$$\text{PDBSM + RAC} \equiv \sum_{i=1}^{m.(n-1)} (RS_{G_i} + WS_{G_i}) + \sum_{i=1}^{m.(n-1)} (WV_{G_i}) + RAC \quad (3)$$

Comparing formulas 1 and 2, it can be easily seen that, in the proposed network setting, the PDBSM protocol using independent certification has lower data consumption as the write values of local fragments are only distributed locally never crossing the long distance links.

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

$$\text{PDBSM+ RAC} < \text{DBSM} \Rightarrow$$
$$RAC < \sum_{i=1}^{m.(n-1)} (RS_{L_i} + WS_{L_i}) + \sum_{i=1}^{m.(n-1)} (WV_{L_i})$$

Finally, formulas 2 and 3, reveal that coordinated certification is preferable when the data required for the RAC does not exceed that for transmitting the read and write sets of partially replicated fragments:

$$\text{PDBSM+ RAC} < \text{PDBSM} \Rightarrow$$
$$RAC < \sum_{i=1}^{m.(n-1)} (RS_{L_i} + WS_{L_i})$$

Until now we have concentrated in data requirements, but we should also pay attention to the expected latencies of the protocols. Every protocol broadcasts the transaction using an atomic broadcast protocol. 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 and in some cases to the designed termination protocol. For example, the PDBSM with RAC protocol regardless of using the RAC implementation offering the lowest cost in terms of latency [**?**], incurs in the additional overhead of the RAC and could present higher latencies.

Furthermore, while observing storage requirements, it is possible to conclude that partial replication, both PDBSM and PDBSM with RAC, outperforms the original DBSM approach. As each replica does not need to be concerned with all the write values, this reduces storage activities and possible bottlenecks. This is an important consideration when considering systems' scalability. The reasoning is that expansions are usually realized with the addition of more database replicas. This should increase the number of simultaneous transactions and therefore the activities of each individual storage, which may not be prepared to handle the additional writes. The same formulas used to analyze the network bandwidth consumption may also be used to determine the storage activities.

The processor does not represent a concern, regarding the extra activities produced by the replication processes, as in the full or partial DBSM "all transaction activities are executed locally". Only upon the commit being issued, the "transaction" updates are propagated to the other replicas. Even the certification procedure, that could increase the processor load, is not a problem, since the read and write sets are upgraded to coarser grains when a threshold is achieved, avoiding a laborious certification and also bandwidth consumption.

4.4 Results

In the experiences we are conducting, we consider a WAN scenario, with 9 replicas. It consists of 3 LANs (with 1Gbps bandwidth and 120μs latency) each with 3 replicas. LANs are connected by a network with a bandwidth of 100Mbps and a latency of 60ms. Each replica corresponds to a dual processor AMD Opteron at 2.4GHz with 4GB of memory, running the Linux Fedora Core 3 Distribution with kernel version 2.6.10. For storage we used a fiber-channel attached box with 4, 36GB SCSI disks in a RAID-5 configuration and the Ext3 file system.

For all the experiments, we varied the total of clients from 20 to 100 and distributed them evenly among the replicas.

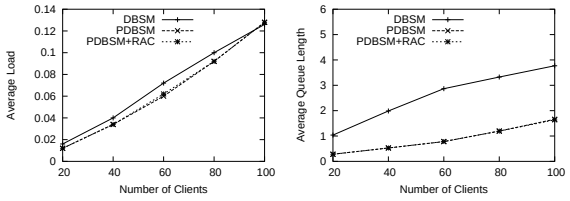
The simulation results are separated in resource consumption (Figure 1), expressed in terms of storage and CPU load, and user results (Figure 2), denoted as abort rate, latency and tpm. The storage load represents amount of data waiting to be served. The CPU load denotes the percentage of the time that the CPU is busy.

As shown in Figure 1, the number of clients used is not enough to fully use the available CPU time. As expected, it makes little difference whether DBSM or PDBSM is used. In contrast, storage is fully occupied and thus increasing queues are formed. Also as expected, the amount of queuing is larger with DBSM. The usage of partial replication results in an obvious economy of resources.

5 Related Work

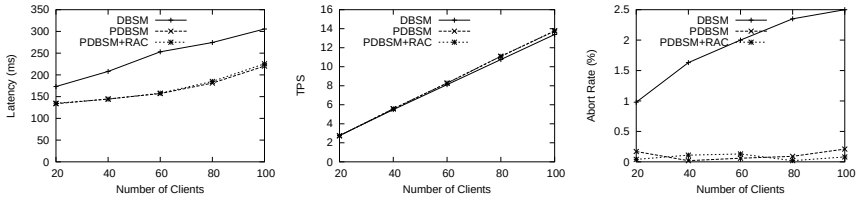
Most of previous work on database replication using group communication [**?**, **?**, **?**, **?**] concentrates on *full replication* strategies. Along with the assumption of the deterministic processing of transactions at every replica, the resulting protocols, characterized as *non voting* [**?**],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

To the best of our knowledge, the works in [**?**, **?**, **?**, **?**] are the only ones to consider partial database replication. Alonso in [**?**] 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 [**?**] 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



(a) CPU (b) Storage

Figure 1. Resource usage.



(a) Latency (b) Throughput (c) Abort rate

Figure 2. Total.

in both approaches presented. In [**?**] the termination protocol fully propagates read and write sets and yet uses a final agreement protocol.

The work of [**?**] 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 replica. Similarly to the PDBSM, transactions are executed optimistically without any coordination between database replicas. Unlike the PDBSM that uses atomic multicast to totally order the certification of transactions, in [**?**], the transaction read and write sets are epidemically propagated and the transaction "certifies" at each replica. By contrast to the PDBSM, a conflict between two transactions dictates the abortion of both.

Except for [**?**], 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 [**?**] 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 [**?**] 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) [**?**] for partial replication. Our main goal is to reduce resource usage associated with 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 restricted to interested replicas. Our experimental results and analysis show that partial replication reduces the used bandwidth, storage activities and the processor load introduced with replication does not represent a concern. The second protocol ensures that both read and write sets as well as updates are all restricted to interested replicas. Our experimental results and analysis show that this approach is still better than full replication. However, the overhead of its required final coordination protocol increases the bandwidth consumption.

These results were obtained with analysis and a realistic traffic pattern, according to the industry standard TPC-C benchmark, in which some frequently used tables were fully replicated to preserve application semantics. The overall evaluation of PDBSM replication using a detailed model of an wide area network and this realistic traffic pattern, is however sufficient to show that partial replication using this approach is useful for large and geographically distributed databases.

References

[1] 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.

[2] F. Pedone, R. Guerraoui and A. Schiper, "The Database State Machine Approach", *J. Distributed and Parallel Databases and Technology*, 2003.

[3] B. Kemme and G. Alonso, "A Suite of Database Replication Protocols based on Group Communication Primitives", in *IEEE Intl. Conf. Distributed Computing Systems*, 1998.

[4] 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.

[5] 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.

[6] 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.

[7] G. Chockler, I. Keidar and R. Vitenberg, "Group communication specifications: a comprehensive study", *ACM Computing Surveys*, vol. 33, n. 4, December 2001.

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

[9] 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.

[10] Transaction Processing Performance Council (TPC), "TPC Benchmark C Standard Specification Revision 5.0", 2001.

[11] T. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems", *Journal of the ACM*, vol. 43, n. 2, 1996.

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

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

[14] 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.

[15] A. Schiper, "Early Consensus in an Asynchronous System with a Weak Failure Detector", *Distributed Computing*, vol. 10, n. 3, 1997.

[16] A. Sousa, J. Pereira, L. Soares, A. Correia Jr., L. Rocha, R. Oliveira and F. Moura, "Testing the Dependability and Performance of Group Communication Based Database Replication Protocols", in *IEEE Intl. Conf. on Dependable Systems and Networks - Performance and Dependability Symposium (DSN-PDS'2005)*, 2005.

[17] J. Cowie, *Scalable Simulation Framework API Reference Manual*, March 1999.

[18] M. Pettersson, "Linux Performance Counters", http://user.it.uu.se/ mikpe/linux/perfctr/, 2004.

[19] A. Correia, A. Menezes and R. Oliveira, "Off-line Test Automation for Database Replication Based on Group Communication", Technical report, Universidade do Minho, 2005.

[20] 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)*, pp. 206–215, Nürnberg, Germany, October 2000, IEEE Computer Society.

[21] 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.

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

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

[24] 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.