Development and evaluation of database replication in ESCADA*

A. Sousa L. Soares A. Correia Jr. F. Moura R. Oliveira
Universidade do Minho

1 Introduction

Software based replication is a highly competitive technique to improve the dependability of database systems. However, the unavoidable trade-off between consistency and performance causes some discredit among database designers with respect to synchronous, strong consistent, replication protocols. This is usually due to performance and scalability problems as classic distributed locking based protocols lead to high resource contention, high transaction latency and high deadlock rates [12]. As a result, commercial database products often privilege asynchronous (or lazy) replication protocols in order to boost performance at the expense of data consistency.

Asynchronous replication is not transparent for the user and therefore cannot be generically applied. Moreover, while strong consistency criteria such as 1-copy-serializability [5] is rigorously defined, relaxed criteria are often ambiguous, hard to formalize, and based on the belief of eventual replica convergence.

To overcome the above problems, a suite of group based communication protocols has emerged and has been the focus of a considerable body of research [3, 21, 26, 14, 19, 11, 15, 22]. Basically, the main and common characteristics of these protocols are the optimistic transaction execution based on deferred updates [5] and the use of total ordered broadcast primitives to enforce a unique sequence of committed transactions.

In some sense, these protocols avoid the efficiency issues of classic distributed locking based protocols by not coordinating the execution of (remote) concurrent transactions and disallow replica divergence of asynchronous replication protocols by aborting transactions that would otherwise violate serializability.

This paper reports our experience on the development and evaluation of group communication based database replication protocols in the ESCADA project, and points out several open issues of current research.

We start the next section by describing the workbench we use for the development and evaluation of database replication protocols. In Section 3, we outline our base protocols and point out their more relevant variations. Section 4 discusses design decisions and identifies open issues. Section 5 concludes the paper.

*Research funded by FCT, ESCADA project (POSI / 33792 / CHS / 2000).

2 Workbench

2.1 Target System

Our target system is modelled as a distributed, reliably connected set of database sites and client sites. All sites communicate by message passing. The system is asynchronous in that there is no bound on process relative speeds, clock drifts, or communication delays.

Sites can fail by crashing. We assume the existence of a failure detector oracle and limits for the number of sites that may fail such that atomic broadcast is implementable [8].

Each database site manages a relational database [17]. Clients submit transactions to database sites in order to be processed. A transaction is a sequence of read and write operations over data items and finishes with a commit or an abort operation. A transaction yields a read set, a write set and the respective write values. The read set is composed by the unique identifiers of the items read. The write set is composed by the unique identifiers of the written items. The write values are the values written to the write set items.

2.2 Simulation Model

From a research perspective, building a real system such as the described for test and evaluation is not feasible. Indeed, it either: *(i)* has too many components to develop that are not the focus of the problem; *(ii)* its use on the evaluation process introduces uncontrollable variables; or *(iii)* its necessary means are not easily available. A native simulation model, even well adjusted to the real one, does not seem to be the best solution either, as there are some aspects of the work that should be measured with a realism that can only be obtained by real implementations. In fact, simulation reveals itself most useful just to recreate components that are not under study, using abstract simulation models. To test and evaluate our protocols in our model, we adopted

the centralized simulation technique [2], and the architecture depicted in Figure 1.

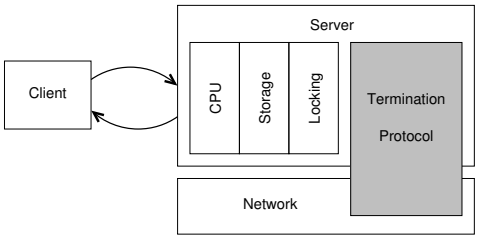

Figure 1. Simulation Architecture.

The network is simulated using the SSFNet network simulation tool [10]. The SSFNet model includes physical network components, such as hosts, links and routers, as well as protocol layers, such as IP, UDP and TCP.

The server component which can handle multiple clients is composed by other components such as CPU and Storage and a Locking policy. Upon receiving a transaction request each operation is scheduled to execute on the corresponding resource. During the simulation run, the usage and length of queues for each resource can be logged and can be used to examine in detail the status of the server.

The replication protocols, that is what we are actually developing and evaluating, are real components.

2.3 Model Instantiation and Validation

In order to use the simulation framework, one needs to perform a framework setup. This setup is based on a validation process to allow a correct model instantiation. In other words, the framework validation enables the definition of a setup which asserts that the simulated system responsiveness is comparable to the real one [25].

Validating the framework is a simple process. It starts by setting up a simple and feasible test case, that is to be recreated in the simulation and in real environments. Once it is defined, real runs are performed on instrumented software. The required instrumented software essentially has logging capabilities that fit the needs imposed by the validation process. In particular, for our work, we use our own customized PostgreSQL database [1] version. The logs produced are used to perform a thorough analysis, from which the correct parameters values, that are needed to carry out the simulation setup, are collected. Setup parameters that cannot be determined using this method, such as for instance storage throughput or latency, are obtained using purpose specific software.

Finally, with the simulation framework already configured, real and simulation runs are compared. The metrics used to perform the comparison relate to performance. For the simulation of database systems, examples of these metrics are: transactions per minute, abort rate and execution latency. Hence, having the model correctly validated, one may start to run simulations varying the intensity of the workload or the replication architecture.

2.4 Application Model

For the application model the TPC benchmarks were chosen. In particular, we use the TPC-C suite. It is the industry standard on-line transaction processing (OLTP) benchmark. It 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. A client can request five different transaction types as follows: New Order, adding a new order to the system (with 44% probability of occurrence); Payment, updating customer's balance, district and warehouse statistics (44%); Order Status, returning a given customer latest order (4%); Delivery, recording the delivery of products (4%); Stock Level, determining the number of recently sold items that have a stock level below a specified threshold (4%).

3 Database Replication Protocols

Our research on group communication based database replication is rooted on the Database State Machine [19] protocol (DBSM). The DBSM was designed with local area networks in mind and applies to fully replicated databases. It has been extended later on [23] to target wide area environments through the exploitation of partial replication (PDBSM). Both DBSM and PDBSM are based on optimistic transaction execution followed by the total ordering of transactions. The differences are to be found in the transaction execution model and in the termination protocol.

3.1 DBSM

The DBSM [19] is a database replication protocol based on group communication that uses the deferred update replication technique [5]. 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, i.e., the transaction enters the committing state. At this point, a termination protocol is started: *(i)* the transaction's relevant information is atomically propagated to all database sites, and *(ii)* each database site certifies the transactions determining its fate: commit or abort. Generally speaking, the certification procedure uses the total order established by the atomic multicast to decide

which transaction must abort when concurrent transactions have conflicting operations,[1] ensuring consistent results.

The design decisions performed when choosing the termination protocol directly affect the overall performance [22]. It is possible to distinguish two situations: independent certification [18] and coordinated certification [15]. In both cases, the protocol atomically multicasts the committing transaction. The differences among the protocols reside in the information being multicast and in the certification procedure executed afterwards.

In the case of independent certification, the exchanged messages include the read and write sets as well as the write values. This allows for each database site to independently certify the transaction and decide whether it commits or aborts.

In the case of the coordinated certification, the exchanged messages just include the write values. This implies that only the database site where the transaction was originated may certify it. Thus, it is responsible for a final atomic commitment protocol.

3.2 Partial-DBSM

Considering database sites spread over a WAN, partial replication can be an attractive solution. Roughly, it explores data locality and eliminates the overhead of transmitting all data across the network. In order to implement partial replication, one needs to determine how data is partitioned and what decisions might affect this process. It might be partitioned (i.e., fragmented) either horizontally or vertically and the whole process is conditioned by: *(i)* the application requirements on data availability, i.e., if data should be available for read and write everywhere or only at some replicas; *(ii)* the local transaction processing, i.e., whether distributed execution is available or not. In the later case, when the distributed execution mechanisms are not available it may be required, due to availability requirements, a large fraction of the database to be fully replicated.

Releasing the assumption that each database site contains a full copy of the database, directly impacts both the execution and the termination protocol [23, 22]. Unlike the DBSM, a given server 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 requires the server to coordinate the distributed processing of the transaction among a set of sites that together contain all the fragments accessed by it. In the DBSM, the whole transaction is relevant to all database sites. In contrast, in a fragmented database this is no longer true. 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.

There are two termination protocols that can be envisaged: one with independent certification and another with coordinated certification. The propagation of the read and write sets determine which protocol must be used. The choice 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 they will enable each site to independently certify 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 judgement and the transaction certification requires a final coordination among all sites [23, 22].

4 Experience, Trade-offs and Open Issues

4.1 In-core or Middleware Approaches

It is possible to implement the previous protocols using either an in-core approach, which requires internal modifications to the database engine, or a middleware approach, which attempts to replace some services that are not provided by the database engine. Roughly, the choice is a trade-off between performance and non-intrusive modifications to the database engine.

For in-core solutions [15, 23], the database engine must expose at least three interfaces which are not usually available. One that provides the write values, the read and write sets produced by the transaction execution. Another one that triggers the termination protocol when receiving the commit request, instead of effectively commiting it. A third one that allows to abort locally executing transactions due to conflicts with a remote transaction.

The middleware solutions appear to avoid intrusive modifications to the database [4, 20, 7] or to circumvent the lack of appropriate interfaces in the database engine. For instance, considering the fact that the read and write sets may not be provided by the database engine, it is necessary to develop an alternative to detect conflicts that is not based on them. In this case, conflicts can be detected by query analysis or transaction's annotations. Specifically, it would be possible to extract the tables and define conflicts based on them. Using such a coarse grain for conflict detection would increase the number of aborts. Approaches based on this model usually proceed as follows [20]:[2] *(i)* atomically multicast the transaction; *(ii)* serialize the execution

[1] Two operations are said to conflict when they belong to concurrent transactions, access the same item and at least one of them is a write operation.

[2] In fact, [20] defines conflict classes that might correspond to tables or even lower grains such as data items. The definition of this classes is however left to the developer.

acquiring locks in the middleware according to the tables; and *(iii)* process the transaction. It would also be possible to extract the predicates available in the queries in order to detect conflicts [13]. Furthermore, the lock acquisition before execution can also be seen as an approach to eliminate the optimistic execution and therefore the aborts [15, 23].

Still based on the assumption that the read and write sets are not provided by the database engine, consider also that write values are not provided or that there is no access to the whole transaction to atomically extract the tables (e.g., interactive transactions). In this scenario, it is necessary to resort to an implementation that for each query guarantees that all the databases with a replica of the referenced tables process them [7, 6]. Unfortunately, the approach requires pre-processing of the queries to extract the tables and the replacement of functions or operations that do not guarantee deterministic results (e.g., date and random functions). Furthermore, since the locks are individually acquired for each one of queries, the approach may generate deadlocks.

If these features are available, the implementation of the termination protocol in middleware can be a good choice, as it allows the combination of different database vendors in a replication scenario. The drawback of the implementation of these features in middleware is that some of the database core functionalities such as the schedule mechanism (i.e., lock acquisition) or the evaluation of queries must be redone in middleware. Another point that discourages this approach is a strong believe that most functionalities required for in-core implementation are internally available in most databases´ core and only need to be exported.

4.2 Optimistic Execution

Upon being successfully certified, a transaction has the guarantee that its outcome will be a commit and applies the updates to the database. During the write process, it has priority over local optimistic transactions that may be executing. If a certified transaction conflicts with a local transaction, the local transaction is aborted and the certified transaction commits. This results in long transactions having higher probability of conflicting with certified transactions.

In TPC-C, as a consequence of its high locality, the number of aborts generated by the optimistic execution is usually not a concern. Indeed, at most 10% of the transactions executed at a database site refer to warehouses stored elsewhere. For that reason, the negative impact of the optimistic execution on the overall performance can be disregarded. However, it is important to notice that this fact is based on the assumption that all the clients of the same warehouse access the same database site and not distinct sites.

Specifically in the case that only read-only transactions are aborted, it is possible to circumvent the problem postponing the updates or using a multiversion database. Unfortunately, the former approach may introduce unpredictable delays on the transactions and the latter approach may not be available in all databases. These problems are identified in [9], which proposes to use the Epsilon Serializability (ESR) in order to stretch the consistency model, improving performance and also reducing the aborts. Generally speaking, it allows results with bounded inconsistencies instead of blocking or aborting the read-only transactions.

Other alternative to avoid aborts exploits the elimination of the optimistic execution. It consists, as explained in Section 4.1, on a lock acquisition prior to execution of the queries in order to serialize the transactions. This approach is divided in two distinct categories. The first allows to atomically acquire the locks of all the tables referenced in a transaction. The second allows to acquire the locks by request. The first was proposed by [20] and preserves the deferred updates. However, it is not clear if the solution is too restrictive due to the serialization prior to the execution. Furthermore, to fully exploit the advantages of this model it is necessary to have sites with distinct access patterns, which means that the possibilities of conflicts are localized in each site. Hence, two questions arise: *(i)* "In this scenario, would not the optimistic execution produce similar results?" — notice that the possibilities of conflicts would be localized in each site and therefore a local strict two-phase locking mechanism, for instance, could control the concurrent transactions; *(ii)* "Is this proposal only suitable for clusters?". The second approach seems to be a regression to the original state machine since each site with a replica of the referenced tables must process the queries.

4.3 Group Communication Protocols

The execution latency of a transaction directly correlates to the probability of aborting it and has a negative impact on the overall performance. In addition, the atomic multicast protocol introduces an increase in latency, specially in wide area networks. However, while it is not possible to reduce this latency, it is possible to reduce its impact on the abort rate and on the overall performance. On sequencer based protocols, one may take advantage of the spontaneous order observed by the sites. If all hosts observe this order with a high degree of probability then they can start using the messages delivered by the spontaneous order to optimistically certify transactions. The decision to commit transactions optimistically certified can only be taken after the order is established and if and only if it matches the optimistic order. Replication protocols exploiting optimistic delivery have been proposed in [16, 23], and an atomic multicast protocol specially tailored for WANs has been proposed in[24].

Recently, probabilistic protocols have emerged as efficient multicast protocols. However, due to their epidemic nature, they can not take advantage of spontaneous order-

ing, rendering optimistic delivery almost useless. Since database replication benefits from optimistic delivery, a hybrid solution is being planned in order to obtain high percentage of spontaneously ordered messages and at the same time an efficient multicast primitive.

4.4 Overall Performance and Resource Usage

Focusing on resource usage, the Partial-DBSM augmented with the distributed execution mechanisms seems to be an attractive solution for WANs, as it presents significant reductions in the required network bandwidth as well as local storage capacity. Once again, a decision must be made on whether it is preferable to send the read and write sets to every replica or to pay the additional latency of an atomic commitment protocol. Considering the fact that on the TPC-C benchmark only 10% of the transactions executed at a database site refer to warehouses stored elsewhere, this strongly suggests the distributed execution mechanism as an attractive solution.

On the other hand, partial replication does not seem to be adequate to LANs and the DBSM is preferable. Results have shown that a replicated database system using DBSM or Partial-DBSM scales, standing side-by-side with an equivalent centralized database, suffering only from a minor increase in CPU usage due to the processing needed regarding communication and termination protocol operations. In addition, CPU usage due to protocol overhead is negligible for a small number of clients, and increases linearly with the increase of the workload [25]. However, in this case the storage bandwidth usage grows identically in all database sites since each replica writes the same values. This must be a concern when scaling up the system, which means that sometimes it may not be sufficient to add another commodity machine. It may also require the upgrade of the storages. This is a delicate issue as costs in improving storage performance, tends to be more expensive than improving the processing capacity.

In clustered environments having a high performance storage per node plays an important role in the cost of the solution. In such a scenario it seems practical to decouple the database front-end, i.e., the processing unit from the database back-end, i.e., the storage, electing some of the replicas as the responsible for effectively commit the transactions to the back-ends.

Having the front-end separated from the back-end, it is easy to perform load balancing when storing information, enabling also database front-ends hot-plugging. As it happens in storage bandwidth, the processing power may prove itself a bottleneck. But then again, given the ability to hot-plug a database front-end and with the help of a fair load-balancing algorithm, this problem may be easily mitigated.

Given the loosely coupled relation between front-ends and back-ends one can easily deploy a replicated system consisting of NxM elements, where N stands for the number of front-ends, and M stands for the number of back-ends.

4.5 Information Propagation

The replication protocols presented raise up an important question about the size of the read sets and as a consequence about the feasibility of transmitting them. Observing the TPC-C specs, except for the Delivery transaction, all read and write sets are relatively small, clearly inferior to 50 data items [22]. However, the Delivery transaction exhibits large read and write sets, around 11200 and 200 items, respectively. The size of the data written by each of the read-write transactions is also relatively small, being around 3.5KB for New Order, 3KB for Payment and 18KB for Delivery, for instance. Analyzing this workload information becomes obvious that special care must be taken about the Delivery transaction, as it may be responsible for some performance problems, due mostly to its large read set size rather than the amount of written data. Inspired by databases locking procedures, which commonly alter the locking granularity when the number of locked items per table exceeds a determined threshold, item locks are upgraded to table locks when a given threshold is reached. Hence, large read sets are reduced to a small set of table locks. This technique greatly reduces the Delivery's read set size making it comparable to the other transactions. However, despite the performance boost regarding latency, this decision should be pondered against the increase on the abort rate caused by it, which is induced by the larger lock granularity.

Other alternative to this problem would be to avoid the transmission of the read set and as a consequence the use of an additional step to propagate the outcome of the transaction as outlined in Section 3.1.

5 Conclusion

The results obtained so far [25], make us strongly believe that group communication based database replication protocols are serious alternatives in terms of performance to commercial solutions based on asynchronous replication and weak consistency models. Resting on rigorous consistency criteria, these protocols not only provide the desired high-availability but also ensure the system's fault tolerance and scale up to the equivalent centralized system.

In this paper we report our experience on the development and evaluation of database replication protocols, discuss the involved trade-offs and present what we think are the main open issues that require further investigation. Implementation issues such as in-core or middleware should be carefully addressed as it is not clear which one to choose

or if a combination of both is the ideal choice. The performance of group communication protocols plays an important role in the overall performance, so efficient group communication protocols for large scale systems would boost the adoption of group based replicated databases in such environments.

References

[1] PostgreSQL. http://www.postgresql.org.

[2] G. Alvarez and F. Cristian. Applying simulation to the design and performance evaluation of fault-tolerant systems. In *IEEE International Symposium on 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, The Hebrew University of Jerusalem, November 1994.

[4] C. Amza, A. Cox, and W. Zwaenepoel. Conflict-Aware Scheduling for Dynamic Content Applications. *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

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

[6] E. Cecchet. C-JDBC Horizontal Scalability design. Technical report, ObjectWeb, 2004.

[7] E. Cecchet, J. Marguerite, and W. Zwaenepoel. C-JDBC:Flexible Database Clustering Middleware. In *USENIX Annual Technical Conference*, 2004.

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

[9] A. Correia Jr., A. Sousa, L. Soares, R. Oliveira, and F. Moura. Revisiting epsilon serializability to improve the database state machine (extended abstract). Technical report, Universidade do Minho, July 2004.

[10] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski. Towards realistic million-node internet simulation. In *Proc. of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, Las Vegas, Nevada, Jun 1999.

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

[12] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. pages 173–182, 1996.

[13] S. Guo, W. Sun, and M. A. Weiss. Solving Satisfiability and Implication Problems in Database Systems. *ACM Trans. Database Syst.*, 1996.

[14] B. Kemme and G. Alonso. A suite of database replication protocols based on communication primitives. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, Amsterdam, The Netherlands, May 1998.

[15] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In *Proceedings of 26th International Conference on Very Large Data Bases (VLDB 2000)*, pages 134–143. Morgan Kaufmann, 2000.

[16] B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proceedings of 19th International Conference on Distributed Computing Systems (ICDCS'99)*, 1999.

[17] P. V. M. Tamer Özsu. *Principles of Distributed Database Systems*. Prentice Hall International, 1999.

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

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

[20] R. J. Peris, M. P. Martínez, B. Kemme, and G. Alonso. Improving the Scalability of Fault-Tolerant Database Clusters. *IEEE International Conference on Distributed Computing Systems*, 2002.

[21] A. Schiper and M. Raynal. From group communication to transactions in distributed systems. 39:84–87, April 1996.

[22] A. Sousa, A. Correia Jr., F. Moura, J. Pereira, and R. Oliveira. Evaluating certification protocols in the partial database state machine. Technical report, Univ. do Minho, 2003.

[23] A. Sousa, F. Pedone, R. Oliveira, and F. Moura. Partial replication in the database state machine. In *IEEE Int'l Symp. Networking Computing and Applications*. IEEE CS, Oct 2001.

[24] 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, Oct. 2002.

[25] A. Sousa, J. Pereira, L. Soares, A. Correia Jr., L. Rocha, R. Oliveira, and F. Moura. Evaluating the performance of the database state machine using realistic simulations. Technical report, Univ. do Minho, 2004.

[26] I. Stanoi, A. Agrawal, and E. Abbadi. Using broadcast primitives in replicated databases (abstract). In *Proceeding of the Sixteen Annual ACM Symposium on Principles of Distributed Computing*, page 283, Santa Barbara, USA, August 1997.