Revisiting 1-Copy Equivalence in Clustered Databases

Rui Oliveira
CCTC/DI
University of Minho
rco@di.uminho.pt

José Pereira
CCTC/DI
University of Minho
jop@di.uminho.pt

Afrânio Correia Jr.
CCTC/DI
University of Minho
acj@di.uminho.pt

Edward Archibald
Emic Networks
ed@emicnetworks.com

ABSTRACT

Recently renewed interest in scalable database systems for shared nothing clusters has been supported by replication protocols based on group communication that are aimed at seamlessly extending the native consistency criteria of centralized database management systems. By using a read-one/write-all-available approach and avoiding the fine-grained synchronization associated with traditional distributed locking, one needs just a single distributed interaction step for each update transaction. Therefore the system can easily be scaled to a large number of replicas, especially, with read intensive loads typical of Web server support environments.

In this paper we point out that 1-copy equivalence for causal consistency, which is subsumed by both serializability and snapshot isolation criteria, depends on basic session guarantees that are costly to ensure in clusters, especially in a multi-tier environment. We then point out a simple solution that guarantees causal consistency in the Database State Machine protocol and evaluate its performance, thus highlighting the cost of seamlessly providing common consistency criteria of centralized databases in a clustered environment.

1. BACKGROUND

A number of group communication based database replication protocols have been proposed, in particular, considering a read-one/write-all-available approach and a single interaction for each update transaction [13, 14, 19, 12]. We focus on protocols that exploit optimistic execution and in particular the Database State Machine (DBSM) [14, 7, 20]. These protocols are multi-master, transactions can be submitted to and executed by several replicas, and follow the passive replication paradigm [5, 9] in which each transaction is executed by one of the replicas and its state changes propagated to the other replicas.

A transaction is immediately executed by the replica to which it is submitted without any a priori coordination. Locally, transactions are synchronized according to the specific concurrency control mechanism of the database engine and thus according to the centralized consistency criteria implemented.

Read-only transactions are simply executed locally at the database to which they are submitted and thus controlled strictly by the local concurrency control mechanism. In contrast, update transactions are not readily committed upon receiving the commit request. Instead, the tuples read (read-set) and written (write-set) are gathered and a termination protocol initiated. The goal of the termination protocol is to decide the order and the outcome of the transaction such that the global correctness criteria is satisfied. This is achieved by establishing a total order position for the transaction and certifying it (i.e., checking for conflicts) against concurrently executed transactions. The certification of a transaction is done by evaluating the intersection of its read-set and write-set (or just its write-set in case of the snapshot-isolation criterion) with the write-sets of concurrent, previously ordered transactions. The fate of a transaction is therefore determined by the termination protocol and a transaction that would locally commit may end up by aborting.

In detail, upon the reception of the commit request for a transaction t, the executing replica atomically multicasts t's id, the version of the database on which t was executed, and t's read-set, write-set and write-values. As soon as t is ordered, each replica is able to certify t on its own. The database version is a counter maintained by the replication protocol that is incremented every time a transaction commits.

To ensure conflict serializability [18], during certification each replica compares its database version with that of t: if they match t commits, otherwise t's read-set and write-set are checked against the write-sets of all transactions committed locally since t's database version. If they do not intersect, t commits, otherwise t aborts. If t commits then its state changes are applied through the execution of a high priority transaction consisting of updates, inserts and deletes according to t's previously multicast write-set and write-values. The high priority of the transaction means that it must be assured of acquiring all the required write locks, possibly aborting any locally executing transactions. The executing replica replies to the client at the end of the transaction.

When considering the snapshot-isolation correctness criterion [3, 1], certification does not need to check read-write conflicts and thus the transactions' read-sets are not required. The DBSM protocol is thus simplified by not propagating the read-sets and using a simpler certification procedure.

At the core of these protocols is a total order (or atomic) multicast primitive provided by a group communication system [6]. This ensures that replicas that remain operational deliver the exact same sequence of messages, and thus, that the deterministic certification procedure produces at all replicas the exact same sequence of outcomes.

2. PROBLEM STATEMENT

Existing proposals adopt the read-one/write-all-available approach

SAC'06 April 23-27, 2006, Dijon, France

for availability and scalability purposes [13, 14, 11, 19, 12, 7, 20]. It offers minimal overhead for read operations and outperforms any other quorum settings in most of the cases [10]. To further improve performance, read-only transactions are not handled by the replication protocol and are not subject to any global synchronization. From a data-centric perspective, the intuition supporting the uncoordinated handling of read-only transactions is that, as long as replica control is done on the transactions' boundaries, reordering a read-only transaction does not impair the serializability of the execution.

However, it can be readily seen that this reordering cannot be applied in general as it might easily contradict the users' local, observable, order of events and violate causal consistency [2] which is subsumed by stronger consistency criteria such as serializability and snapshot-isolation [1].

Roughly, causal consistency means that any two causally related write operations must be seen by all server replicas in the same order. As demonstrated by Brzeziński et al. [4] causal consistency requires four basic session guarantees [16] to be preserved: *read your writes*, *monotonic reads*, *monotonic writes* and *writes follow reads*. From these, the first two are endangered by the uncoordinated handling of read-only transactions. The *read your writes* condition ensures that a read operation is executed by a database replica that has performed all writes previously issued by the requesting client. *Monotonic reads* ensures that read operations are executed by database replicas that have performed all writes seen by previous reads of the requesting client.

To see how the DBSM protocol fails to guarantee both conditions above consider the following examples. Let db_1 and db_2 be two database replicas initially synchronized. Let $T_w = w(x)v$ be a transaction executing at replica db_1. If the same user of T_w afterwards issues transaction $T_{ra} = r(x)$ and T_{ra} is handled by db_2 it is not guaranteed that T_{ra} returns v. The reason is that since T_{ra} is read-only and thus was handled at replica db_2 without any prior synchronization with db_1, it may happen that when it is executed at db_2 the replica is still not updated and therefore not in sync with db_1. The user therefore misses her writes.

Suppose now that T_w and T_{ra}, not necessarily from the same user, are handled in this order by the same replica db_1. Clearly, T_{ra} returns v as the value of x. If the user of T_{ra} afterwards issues a transaction $T_{rb} = r(x)$ that happens to be handled by replica db_2, it is not guaranteed that T_{rb} returns v. The reason is the same as above but now also the *monotonic reads* condition is not guaranteed.

Notice that the above phenomena is of the sole responsibility of the replica control protocol and independent of the centralized consistency criteria of the replicas. Since the problem may only occur when transactions from the same client are handled by different replicas, a general workaround, would be to simply have all requests of a client being sent to the same replica. This could either be done by the replication protocol itself or, even more simply, delegating it to a load-balancing layer preserving client/replica affinity.

Unfortunately the assumption of client/replica affinity is increasingly harder to ensure in the typical usage scenario for database clusters which are multi-tier systems. In contrast with traditional systems in which each user maintains a private session and a private database connection, it is now common that clients connect to an application server which maintains a pool of database connections and dynamically dispatches requests on behalf of multiple clients. To optimize performance, such connections should have been evenly distributed across available replicas by a load balancer. Additionally, a caching layer maintained within the application server can be used by multiple clients. Finally, the same end-user might even use multiple concurrent connections to the application server that cannot be easily tracked to a single entity. It is therefore unfeasible to assume that connections to different replicas are unrelated.

3. PROPOSED SOLUTION

We modified the DBSM replication protocol [14] so that read-only transactions receive appropriate synchronization to preserve the 1-copy equivalence of causal consistency. We explored two different approaches. In the first, the protocol simply handles read-only and update transactions in the same way. Read-only transactions, upon the reception of the commit request, are now also totally ordered with respect to all other transactions and subject to certification. Nevertheless, a read-only transaction is only certified at its executing replica. Since such a transaction does not change the database and has no influence on the certification of subsequent transactions, there is no point in certifying it at the other replicas. In this approach, with the optimistic execution and later certification of read-only transactions, the examples of Sect. 2 are prevented by aborting the read-only transaction T_{ra} in the former and T_{rb} in the latter. From the first example, T_{ra} is ordered after T_w. When T_{ra} is requested to commit, it is certified against the write-sets of transactions that commited after T_{ra} started executing. If T_{ra} executed after the local commit of T_w then T_{ra} commits, otherwise since its read-set intersects with T_w's write-set, T_{ra} aborts. The reasoning for the *monotonic reads* example is similar.

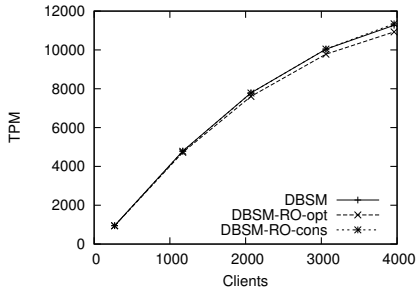
Our second approach handles read-only transactions differently, their execution is not optimistic. As soon a read-only transaction begins its id is atomically multicast to be totally ordered. It is then executed when all update transactions ordered before it are commited at the executing replica. Considering again the examples of Sect. 2, in this variation of the DBSM protocol, the read-only transactions T_{ra} and T_{rb} would be both ordered after T_w in both cases. T_{ra} in first example (T_{rb} in the second) would be executed at db_2 only after the commit of T_w at db_2.

The need to totally order all read-only transactions translates, as expected, into increased latency in both protocols. The two approaches offer however a trade-off between response time at the expense of some aborts (leaving to the user the decision of resubmitting the transaction) and abort-free executions of read-only transactions with increased latency.

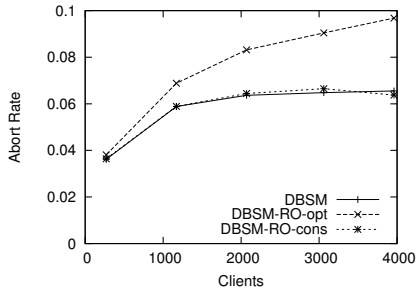
4. EVALUATION

To evaluate the cost of the changes made to the DBSM protocol we tested both variations under the workload specified by the TPC-C benchmark [17]. We used a centralized, hybrid simulation environment that combines real software components with simulated hardware, software and environment components to model a distributed system [15]. This allows us to set up and run multiple realistic tests with slight variations of configuration parameters that would otherwise be impractical to perform, specially if one considers a large number of replicas. The key components, the replication and the group communication protocols, are real implementations while both the database engine and the network are simulated.

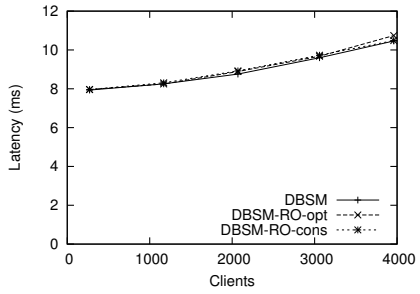
The simulated database server handles multiple clients and is modeled as a scheduler and a collection of resources, such as storage and CPUs, and a concurrency control module. The database implements a multi-version concurrency control. Each transaction is modeled as a sequence of operations: i) fetch 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



(a)

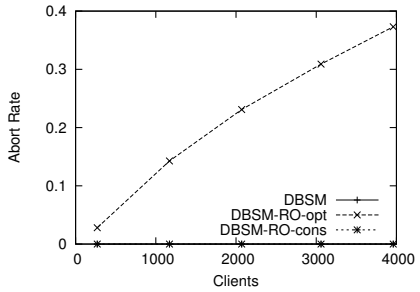


(b)

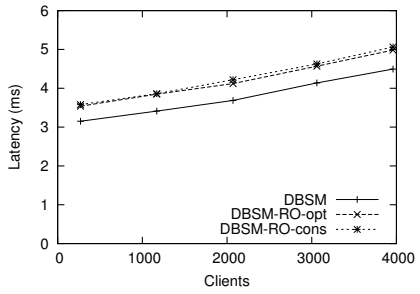


(c)

Figure 1: Overall performance measurements.



(a)



(b)

Figure 2: Performance of read-only transactions.

corresponding resource. The processing time of each operation is previously obtained by profiling a real database server.[1] A database client is attached to a database server and produces a stream of transaction requests. There are five different types of transactions: *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. 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.

We consider a database cluster with 9 replicas connected by a network with a bandwidth of 1Gbps and a latency of 120μs. Each replica corresponds to a dual processor AMD Opteron at 2.4GHz with 4GB of memory, running Linux. For all the experiments that follow, we varied the total of clients from 270 to 3960 and distributed them evenly among the replicas.

Fig. 1 shows the overall performance of the original DBSM protocol (i.e. without certifying read-only transactions) — DBSM, the variation that certifies read-only transactions executed optimistically — DBSM-RO-opt, and the one that executes read-only transactions conservatively — DBSM-RO-cons. Fig. 2 shows the latency and abort rate of the read-only transactions according to these protocols. Fig. 3 shows specifically the stock level transactions' behavior.

According to Fig. 1(a) and 1(c), the impact on throughput and latency imposed by the DBSM-RO-opt and the DBSM-RO-cons is negligible. However, Fig. 1(b) shows that the DBSM-RO-opt for 3960 clients increases the abort rate from approximately 6% to 9%.

However, the apparently modest increase in abort rate with the DBSM-RO-opt algorithm hides an unacceptable abort rate of 37% for the read-only transactions (Fig. 2(a)). Regarding read-only transactions' latency, Fig. 2(b) shows that it is increased by an average of approximately 0.5 ms in the DBSM-RO-opt and 0.6 ms in the DBSM-RO-cons. Although, as expected, the DBSM-RO-cons imposes a higher overhead in terms of latency when compared to the DBSM-RO-opt, the difference is almost negligible.

Examining aborted transactions in more detail, the new aborts are due to the serialization of the stock level transaction, which has a high probability of conflicts when compared to the order status transaction. Basically, the former access tables that have a low number of tuples when compared to the latter thus increasing the likelihood of conflicts. Fig. 3(a) shows that for 3960 clients approximately 74% of the stock level transactions are aborted in the DBMS-RO-opts. The latency, as shown in Fig. 3(b), follows the same pattern presented in the later set of figures. The latency of the stock level is approximately increased by 0.5 ms in the DBSM-RO-opt and 0.6 ms in DBSM-RO-cons.

[1]We used a profiled version of PostgreSQL 7.4.6 under the TPC-C workload.

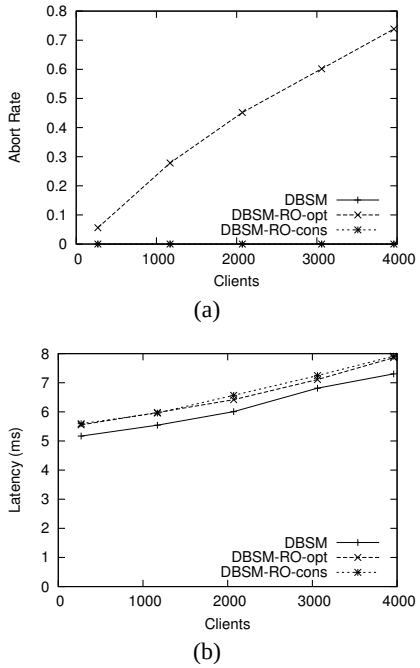


Figure 3: Performance of stock-level transactions (read-only).

5. DISCUSSION

Besides being inherently fault-tolerant, database replication protocols based on group communication have been pointed out as a good choice as the basis for clustered database systems due to their performance and scalability as well as the ability to seamlessly provide the 1-copy equivalence for the consistency criteria of centralized databases.

In this paper we point out that an often unspoken assumption required for 1-copy equivalence, that clients issue requests strictly to a single replica and do not otherwise share information, strongly contradicts the actual usage of database clustering for performance. In detail, in common multi-tier applications multiple clients share cached information in application server tiers and share a connection pool to database servers. Such connections must be directed to multiple database replicas to take advantage of parallel processing. In this setting, the replicated system can violate even the weak causal consistency criteria.

Although we use the Database State Machine [14] protocol as an example, the same problem exists in all optimistic approaches, whether providing conflict serializability or snapshot isolation, and even on conservative approaches that serialize transactions prior to execution [13]. This follows from all such protocols allowing deferred updates and at the same time not ordering read-only transactions.

The proposed solution is thus to order read-only transactions, either after executing them optimistically or conservatively, prior to their execution. By using the industry standard TPC-C benchmark workload, we show that conservative ordering is preferable in a cluster in which the small latency increase offsets the large number of transactions aborted due to certification in the optimistic protocol. Alternatively, adopting a relaxed consistency criteria such as Generalized Snapshot Isolation [8] that encompasses the use of stale snapshots would obviate the synchronization of read-only transactions.

6. REFERENCES

[1] A. Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. PhD thesis, MIT, Mar. 1999.

[2] M. Ahamad, G. Neiger, P. Kohli, J. Burns, and P. Hutto. Causal memory: definitions, implementation and programming. *Distributed Computing*, 9(1), Sept. 1995.

[3] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *ACM SIGMOD Int. Conf. on Management of Data*, 1995.

[4] J. Brzeziński, C. Sobaniec, and D. Wawrzyniak. From session causality to causal consistency. In *Euromicro Conf. on Parallel, Distributed and Network-Based Processing*, 2004.

[5] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. The primary-backup approach. In S. Mullender, editor, *Distributed Systems*, chapter 8. Addison Wesley, 1993.

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

[7] A. Correia Jr., A. Sousa, L. Soares, J. Pereira, F. Moura, and R. Oliveira. Group-based replication of on-line transaction processing servers. In *2nd Latin-American Symposium on Dependable Computing*, 2005.

[8] S. Elnikety, F. Pedone, and W. Zwaenepoel. Database replication using generalized snapshot isolation. In *Proc. IEEE Int'l Symp. Reliable Distributed Systems (SRDS)*, 2005.

[9] R. Guerraoui and A. Schiper. Software-based replication for fault tolerance. *IEEE Computer*, 30(4), Apr. 1997.

[10] R. Jiménez-Peris, M. Patiño-Martinez, G. Alonso, and B. Kemme. Are quorums an alternative for data replication? *ACM Trans. on Database Systems*, 28(3), Sept. 2003.

[11] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *Intl. Conf. Very Large Data Bases (VLDB)*, pages 134–143, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[12] Y. Lin, B. Kemme, M. Patiño-Martinez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *ACM SIGMOD Int. Conf. on Management of Data*, 2005.

[13] M. Patiño-Martinez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *Intl. Conf. Distributed Computing (DISC)*, pages 315–329, London, UK, 2000. Springer-Verlag.

[14] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *J. Distributed and Parallel Databases and Technology*, 2003.

[15] 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. Dependable Systems and Networks - Performance and Dependability Symp. (DSN-PDS'2005)*, 2005.

[16] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch. Session guarantees for weakly consistent replicated data. In *Intl.Conf. on Parallel and Distributed Informations Systems*, 1994.

[17] T. P. P. C. (TPC). TPC Benchmark^{TM} C standard specification revision 5.0, Feb. 2001.

[18] G. Weikum and G. Vossen. *Transactional Information Systems*. Morgan Kaufmann, 2002.

[19] S. Wu and B. Kemme. Postgres-r(si): Combining replica control with concurrency control based on snapshot isolation. In *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, pages 422–433, April 2005.

[20] V. Zuikeviciute and F. Pedone. Revisiting the database state machine approach. In *VLDB Wks. on Design, Implementation and Deployment of Database Replication*, 2005.