Revisiting the Database State Machine Approach

Vaidė Zuikevičiūtė

Università della Svizzera Italiana (USI)
Via Lambertenghi 10A
CH-6904 Lugano
Switzerland
vaide.zuikeviciute@lu.unisi.ch

Fernando Pedone

Università della Svizzera Italiana (USI)
Via Lambertenghi 10A
CH-6904 Lugano
Switzerland
fernando.pedone@unisi.ch

Abstract

The Database State Machine (DBSM) is a replication mechanism for clusters of database servers. Read-only and update transactions are executed locally, but during commit, update transactions execution outcome is broadcast to all the servers for certification. The main DBSM's weakness lies in its dependency on transaction readsets, needed for certification. This paper presents a technique to bypass the extraction and propagation of readsets. Our approach does not incur any communication overhead and still guarantees that transactions are serializable.

1 Introduction

Replication is an area of interest to both distributed systems and databases: in database systems replication is done mainly for performance and availability, while in distributed systems mainly for fault tolerance. The synergy between these two disciplines offers an opportunity for the emergence of database replication protocols based on group communication primitives.

This paper focuses on the Database State Machine (DBSM) approach [20]. The DBSM is based on deferred update replication, implemented as a state machine. Read-only transactions are processed locally at some database replica; update transactions do not require any synchronization between replicas until commit time. During commit, the transaction's updates, readsets, and writesets are broadcast to all servers for certification. To ensure that each replica converges to the same state, each server has to reach the same decision when certifying transactions and guarantee that conflicting transactions are applied to the database in the same order. These requirements are enforced by an atomic broadcast primitive and a deterministic certification test.

The DBSM has several advantages when compared to existing replication schemes. In contrast to lazy replication techniques, the DBSM provides strong consistency (i.e., serializability) and fault tolerance. When compared with primary-backup replication, it allows transaction execution to be done in parallel on several replicas, which is ideal for workloads populated by a large number of non-conflicting update transactions. By avoiding distributed locking used in synchronous replication, the DBSM scales to a larger number of nodes. Finally, when compared to active replication, it allows better usage of resources because each transaction is executed by a single node.

The main weakness of the DBSM lies in its dependency on transaction readsets, needed for certification. Extracting readsets usually implies changing the database internals or parsing SQL statements outside the database, both undesirable solutions due to portability, complexity, and performance reasons. On the other hand, extracting writesets is less of a problem: writesets tend to be much smaller than readsets and can be obtained during transaction processing (e.g., using triggers). This paper extends the original DBSM to avoid the need of readsets during certification. Our approach has no communication and consistency penalties: termination still relies on a single atomic broadcast and the execution is still serializable. Moreover, in most cases of practical interest, the price to pay is a few additional aborted transactions.

The remainder of the paper is organized as follows: Section 2 introduces our system model and some definitions. Section 3 explains how to avoid readsets during certification. Section 4 presents some preliminary performance results, and Section 5 discusses related work. Section 6 summarizes the proposed ideas and gives an overview of future refinements.

2 System model and definitions

In this section we present the DBSM and the two concepts it relies upon: state machine and group communication. The state machine approach delineates the replication strategy. Group communication primitives constitute a sufficient mechanism to implement a state machine.

2.1 Database replication

We consider a system $\Sigma = \{s_1, s_2, ..., s_n\}$ of database sites. Sites communicate with each other through atomic broadcast, built on top of message passing. Replicas fail independently and only by crashing (i.e., we exclude Byzantine failures). Database sites may eventually recover after a crash.

Each database site plays the role of a replica manager and each has a full copy of the database. Transactions are locally executed according to strict two-phase locking (2PL). Our consistency criteria is one-copy serializability [18].

Transactions are sequences of read and write operations followed by a commit or abort operation. A transaction is called a query (or read-only) if it does not contain any write operations; otherwise it is called an update transaction.

2.2 State machine replication

The state machine approach is a non-centralized replication technique. Its key concept is that all replicas receive and process the same sequence of requests in the same order. Consistency is guaranteed if replicas behave deterministically, that is, when provided with the same input (e.g., a request) each replica will produce the same output (e.g., state change).

The way requests are disseminated among replicas can be decomposed into two requirements [23]:

1. **Agreement.** Every non-faulty replica receives every request.
2. **Order.** If a replica processes request req_1 before req_2, then no replica processes req_2 before req_1.

Notice that the DBSM does not require the execution of transaction to be deterministic; only the certification test is implemented as a state machine.

2.3 Atomic broadcast communication

In order to satisfy the above mentioned state machine requirements, database sites interact by means of atomic broadcast, a group communication abstraction. Atomic broadcast guarantees the following properties:

1. **Agreement.** If a site delivers a message m then every site delivers m.
2. **Order.** No two sites deliver any two messages in different orders.
3. **Termination.** If a site broadcasts message m and does not fail, then every site eventually delivers m.

Several atomic broadcast algorithms exist in the literature [6]. Our experiments (see Section 4) are based on a highly efficient Paxos algorithm [15].

2.4 The Database State Machine approach

The Database State Machine is based on deferred update replication. During transaction execution there is no interaction between replicas. When an update transaction is ready to be committed, its updates (e.g., redo logs), readsets, and writesets are atomically broadcast to all replicas. All sites receive the same sequence of requests in the same order and certify them deterministically. The certification procedure ensures that committing transactions do not conflict with concurrent already committed transactions.

The notion of conflicting concurrent transactions is based on the *precedence relation*, denoted by $t_j \rightarrow t_i$, and defined next.

- If t_i and t_j execute on the same replica s_i, then $t_j \rightarrow t_i$ only if t_j enters the committing state at s_i before t_i enters the committing state at s_i.
- If t_i and t_j execute on different replicas s_i and s_j, respectivelly, then $t_j \rightarrow t_i$ only if t_j is committed at s_i before t_i enters the committing state at s_i.

Two operations *conflict* if they are issued by different transactions, access the same data item and at least one of them is a write. Finally, a transaction t_j *conflicts with* t_i if they have conflicting operations and t_j does not precede t_i.

During processing, transactions pass through some well-defined states:

1. *Executing state.* In this state transaction t_i is locally executed at site s_i according to strict 2PL.
2. *Committing state.* Read-only transactions commit immediately upon request. If t_i is an update transaction, it enters the committing state and s_i starts the termination protocol for t_i: t_i's updates, readsets, and writesets are broadcast to all replicas. Upon delivering this message, each database site s_i certifies t_i. Transaction t_i passes the test at s_i if the following condition holds:

$$\left[\begin{array}{l} \forall t_j \text{ committed at } s_i : \\ t_j \rightarrow t_i \vee (writesets(t_j) \cap readsets(t_i) = \emptyset) \end{array} \right]$$

   Atomic broadcast is used to ensure that the sequence of transactions certified by each replica is the same, thus consistency is guaranteed.

3. *Committed/Aborted state.* If t_i passes the certification test, its updates are applied to the database and t_i passes to the committed state. Transactions in the executing state at s_j holding locks on data items updated by t_i are aborted.

3 DBSM*: refining the DBSM

In the original DBSM, readsets of update transactions need to be broadcast to all sites for certification. Although storing and transmitting readsets are sources of overhead, extracting them from transactions is a more serious problem since it usually implies accessing the database internals or parsing SQL statements outside the database. For the sake of portability, simplicity, and efficiency, certification should be "readsets free."

3.1 Readsets-free certification

The basic idea of the DBSM remains the same: transactions are executed locally according to strict 2PL. In contrast to the original DBSM, when an update transaction requests a commit, only its updates and writesets are broadcast to the other sites. Certification checks whether the writesets of concurrent transactions intersect; if they do, the transaction is aborted. Transaction t_i passes certification at s_i if the following condition holds:

$$\left[\begin{array}{l} \forall t_j \text{ committed at } s_i : \\ t_j \rightarrow t_i \vee (writesets(t_j) \cap writesets(t_i) = \emptyset) \end{array} \right]$$

Does such a certification test ensure one copy serializability? Transactions executing at the same site are serializable, but the certification test does not ensure serializability of global transactions: not all the serialization anomalies are avoided in the execution [9]. For instance, it does not avoid the *write-skew* anomaly:

$$r_i[x], r_i[y] \ ... \ r_j[x], r_j[y], w_j[x], c_j \ ... \ w_i[y], c_i$$

In the above example t_i and t_j are executed concurrently at different sites, t_i reads x and y, t_j reads the same data items, writes x and tries to commit. Then t_i writes y and tries to commit. Both transactions pass the certification test because their writesets do not intersect, however the execution is not serializable (i.e., no serial execution is equivalent to it).

3.2 Snapshot Isolated DBSM*

Snapshot isolation (SI) is a multi-version concurrency control algorithm introduced in [9]. SI does not provide serializability, but is still attractive and used in commercial and open-source database engines, such as Oracle and PostgreSQL. Under SI a transaction t_i sees the database state produced by all the transactions that committed before t_i started. Thus if t_i and t_j are concurrent, neither will see the effects of the other. According to the *first-committer-wins* rule, t_i will successfully commit only if no other concurrent transaction t_j that has already committed writes to data items that t_i intends to write.

Although specific workloads will not be serializable under SI, such cases seem to be rare in practice. Fairly complex transaction mixes, such as the TPC-C benchmark, are serializable under SI. Moreover, there are different ways to achieve serializability from SI [1, 2].

Transactions executing in the same site in the DBSM* are snapshot isolated. This follows from the fact that such transactions are serializable, and serializability is stronger than SI [9]. Are global transactions also snapshot isolated in the DBSM*? It turns out that the answer to this question is yes. We provide only an informal argument here. First, notice that any two concurrent transactions executing in different sites are isolated from one another in the DBSM*: one transaction does not see any changes performed by the other (before commit). Second, the DBSM*'s certification test provides the first-committer-wins behavior of SI since the first transaction to be delivered for certification commits and later transactions abort.

3.3 One-copy serializable DBSM*

The DBSM*, as well as the original DBSM, has an interesting property: if all transaction requests are submitted to the same replica, the DBSM* will behave as primary-backup replication [26]. Since all transactions would then be processed according to 2PL at the primary site, 1SR would be ensured. Therefore, ensuring 1SR in the DBSM* would be a matter of carefully scheduling update transactions to some selected database site; read-only transactions could still be executed at any replica. However, for load-balancing and availability reasons, localizing the execution of update transactions in the same site may not be such a good idea.

In [1] it was proved that two transactions executing concurrently under SI produce serializable histories if they are *interference-free*, or their writesets intersect. Two transactions are interference free if one does not read what is written by the other (notice that this is precisely what the original DBSM certification test guarantees). Therefore, two transactions concurrently executed at different sites in the DBSM* are serializable if their writesets intersect or one does not read what is written by the other. Since this may not hold for each pair of update transactions, the authors in [2] suggested conflict materialization techniques as a way to guarantee serializable histories of dangerous transactions executed under SI.

Following these ideas, we describe next our technique, which guarantees 1SR with no communication overhead w.r.t. the original DBSM. Briefly, the mechanism works as follows:

1. The database is logically divided into a number of disjoint sets (according to tables, rows, etc), each one under the responsibility of a different replica, and extended with a control table containing one dummy row per logical set. This control table is used for conflict materialization. Note that each replica still stores a full copy of the database.

2. Each replica is responsible for processing update transactions that access data items in its assigned logical set.Transactions that only access data items in a single logical set and execute at the corresponding replica (we call them "local") are serialized with other transactions of the same type by the 2PL scheduler on the server where they execute.

3. Update transactions that access data items in more than a logical set should execute on a server responsible for one of these logical sets. We call such transactions "complex". Complex transactions are serialized with other transactions updating data items in intersecting logical sets by the certification test. But the certification test cannot serialize them with interfering transactions executing at different servers.

4. To ensure 1SR update transactions that read data items in a logical set belonging to the remote replica are extended with update statements for dummy rows corresponding to each remote logical set read.This can be done when the application requests the transaction commit. Dummy rows are constructed in such a way to materialize write-write conflicts between complex or local transactions that access data items in the same logical set.Therefore, if t_i executes at s_i and one of t_i's operations reads a data item that belongs to s_j's logical set, a dummy write for s_j logical set is added to t_i. This ensures that if t_i executes concurrently with some transaction in s_j, complex or not, only one transaction will pass the certification test.

5. Read-only transactions can be executed at any database site independently of the data items accessed.

Correctness is a consequence of the fact that read-only and version-order dependencies in multiversion serialization graph follow the delivery order of committed transactions. We prove this formally in an extended version of this document.

Abort rates can be reduced in the above scheme if the division of the database into logical sets takes the workload into account. For example, a criterion for defining logical sets could be the tables accessed by the transactions. Moreover, notice that we do not have to know exactly which data items are accessed by a transaction to schedule it to its correct server; only its logical sets have to be known (e.g., which tables are accessed by the transaction).

The issue of transaction scheduling is orthogonal to the described work and a detailed account of it is beyond the scope of this paper. Whatever mechanism is used, it should obviously be aware of the division of the database into logical sets and schedule update transactions accordingly. Nevertheless any transaction can be executed at any database site as long as a corresponding dummy writes for remote logical sets read are added to materialize the conflict.

4 Performance results

In this section we evaluate the impact of the modifications proposed on the abort rate of the DBSM.

4.1 Simulation model

All experiments were performed using a discrete-event simulation model written in C++ based on the C-SIM library. Every server is modelled as a processor with some data disks and a log disk as local resources. The network is modeled as a common resource shared by all database sites. Communication delays between replicas are based on experiments measuring Paxos in a real network.

The workload contains 50% of update and 50% of read-only transactions. The number of operations within a transaction varies from 5 to 15. Each database replica contains 2000 data items. We assume that access to data items is uniform – there are no hotspots.

4.2 Experiments

We are interested here in understanding the implications of conflict materialization on the abort rate of the DBSM*. Conflict materialization introduces additional aborts to the DBSM* whenever a conflict arises between concurrently executing transactions. Such aborts are essential to avoid non-serializable executions.

In the graphs presented next, each plotted point was determined from a sequence of simulations, each containing 100000 submitted transactions. In order to remove initial transients, only after the first 1000 transactions had been submitted the statistics started to be gathered.

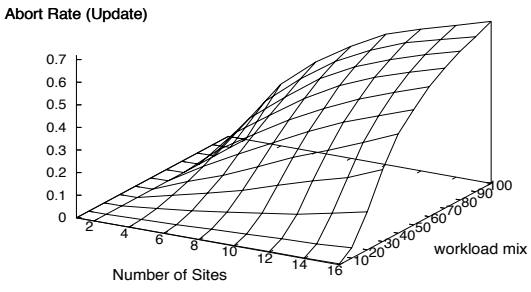

Figure 1: DBSM*'s abort rate (updates only)

Figure 1 presents the abort rate of update transactions when the number of database sites and the percentage of complex update transactions increases. In a workload mix of 100%, all transactions are complex. Obviously, the abort rate increases with the number of complex transactions and the number of sites. More interesting, workload mixes with few complex transactions (0%–15%) tend to scale very well with the number of sites. This is particularly important since in practice, the percentage of complex transactions is fairly low. For example, in the heaviest transaction mix of the TPC-W benchmark (the *ordering scenario*, with the highest number of update transactions) the percentage of complex transactions varies from 10%–15% of all update transactions. Moreover, TPC-C is serializable under SI, and therefore no conflict materialization is needed.

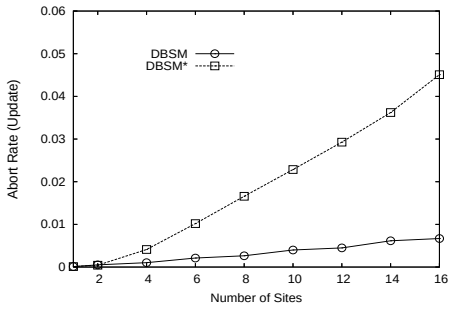

Figure 2: DBSM* vs. DBSM (updates only)

In Figure 2 we compare the abort rate of update transactions executing in the original DBSM and the DBSM*. In the experiments, the workload mix contains 10% of complex update transactions. Although the DBSM aborts fewer transactions than the DBSM*, both abort less than 5% of update transactions.

5 Related Work

Several works have proposed database replication centered on group communication. Among them Agrawal et al. present a family of replica management protocols that exploit the properties of atomic broadcast and are based on immediate and deferred replication [5]. Deferred replication based on group communication was also proposed in [8].

A suite of eager, update everywhere replication protocols is introduced in [12]. The authors take advantage of the different levels of isolation provided by databases to relax the consistency of the protocols. In [4] Kemme and Alonso introduce Postgres-R, a replication mechanism implemented within PostgreSQL. Snapshot isolation is further investigated in [27, 16].

In [21] the authors present Ganymed, a middleware-based replication solution. The main idea behind Ganymed is a separation between updates and read-only transactions: updates are handled by a master replica and lazily propagated to the slaves, where queries are processed.

Clustered JDBC (C-JDBC) [7] is another middleware-based replication solution. C-JDBC supports both partial and full replication. The approach consists in hiding a lot of database complexity in the C-JDBC layer, outside the database. Extending JDBC with a primary-backup technique was proposed in [19].

Some works have concentrated on reducing the communication overhead of group communication by overlapping transaction processing with message ordering [13, 14, 17]. While NODO [17] protocol requires transactions scheduling to be known in advance, that is, before the transactions execution, in DBSM* scheduling is more like a hint: 1SR is ensured even if transactions are scheduled independently of data items accessed.

Amir et al. [3] deal with replication in wide area networks. An active replication architecture is introduced by taking advantage of the Spread Group Communication Toolkit. Spread provides atomic broadcast and deals with network partitions. In [22] other two replication protocols based on atomic multicast for large-scale networks are presented.

The original DBSM has been previously extended in two directions. Sousa et al. investigate the use of partial replication in the DBSM [25]. In [11] the authors relax the consistency criteria of the DBSM with Epsilon Serializability.

A number of works have compared the performance of group-based database replication. In [10] Holliday et al. use simulation to evaluate a set of four abstract replication protocols based on atomic broadcast. The authors conclude that single broadcast transaction protocol is the one that allows better performance by avoiding duplicate execution and blocking. This protocol abstracts the DBSM. Another recent work [24] evaluates the original DBSM approach, where a real implementation of DBSM's certification test and communication protocols is used. All the results confirm the usefulness of the approach.

6 Conclusion

The Database State Machine is a simple approach to handle database replication. This paper addresses one of its main weaknesses: the dependency on transaction readsets for certification. The conflict materialization technique adopted for DBSM* does that without sacrificing one copy serializability and increasing communication overhead. Depending on the workload, transactions with dangerous structures can be forced to execute on the same site or at different replicas, if an artificial update on dummy row is introduced in the transaction. As future work we plan to prototype the DBSM* and optimize our conflict materialization algorithm.

7 Acknowledgments

We would like to thank the anonymous reviewers for their remarks which helped us to improve the paper.

References

[1] A.Fekete. Serialisability and snapshot isolation. In *Proceedings of the Australian Database Conference, Auckland, New Zealand*, January 1999.

[2] A.Fekete, D.Liarokapis, E.O'Neil, P.O'Neil, and D.Shasha. Making snapshot isolation serializable. Unpublished manuscript, available at http://www.cs.umb.edu/isotest.

[3] Y. Amir and C. Tutu. From total order to database replication. In *Proceedings of the International Conference on Dependable Systems and Networks*, 2002.

[4] B.Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000.

[5] D.Agrawal, G.Alonso, A.El Abbadi, and I.Stanoi. Exploiting atomic broadcast in replicated databases. In *Proceedings of the 3th International Euro-Par Conference on Parallel Processing*, 1997.

[6] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, 2004.

[7] E.Cecchet, J.Marguerite, and W.Zwaenepoel. C-JDBC: Flexible database clustering middleware. In *Proceedings of USENIX Annual Technical Conference, Freenix track*, 2004.

[8] F.Pedone, R.Guerraoui, and A.Schiper. Transaction reordering in replicated databases. In *Proceedings of the 16th IEEE Symposium on Reliable Distributed Systems*, 1997.

[9] H.Berenson, P.Bernstein, J.Gray, J.Melton, E. O'Neil, and P.O'Neil. A critique of ANSI SQL isolation levels. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1995.

[10] J. Holliday, D.Agrawal, and A. E. Abbadi. The performance of database replication with group multicast. In *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing*, 1999.

[11] A. Correia Jr., A. Sousa, L. Soares, F. Moura, and R. Oliveira. Revisiting epsilon serializabilty to improve the database state machine (extended abstract). In *Proceedings of the Workshop on Dependable Distributed Data Management, SRDS*, 2004.

[12] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *Proceedings of the International Conference on Distributed Computing Systems*, 1998.

[13] 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*, 1999.

[14] B. Kemme, F. Pedone, G. Alonso, A. Schiper, and M. Wiesmann. Using optimistic atomic broadcast in transaction processing systems. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):1018–1032, July 2003.

[15] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[16] Y. Lin, B. Kemme, M. Patino-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of data*, 2005.

[17] M. Patino-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *Proceedings of the 14th International Conference on Distributed Computing*, 2000.

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

[19] F. Pedone and S. Frolund. Pronto: A fast failover protocol for off-the-shelf commercial databases. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, 2000.

[20] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Journal of Distributed and Parallel Databases and Technology*, 14:71–98, 2002.

[21] C. Plattner and G. Alonso. Ganymed: scalable replication for transactional web applications. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, 2004.

[22] L. Rodrigues, H. Miranda, R. Almeida, J. Martins, and P. Vicente. The GlobData fault-tolerant replicated distributed object database. In *Proceedings of the 1st Eurasian Conference on Advances in Information and Communication Technology*, 2002.

[23] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[24] 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 *Proceedings of IEEE International Conference on Dependable Systems and Networks - Performance and Dependability Symposium*, 2005.

[25] A. Sousa, F. Pedone, F. Moura, and R. Oliveira. Partial replication in the database state machine. In *Proceedings of the IEEE International Symposium on Network Computing and Applications*, 2001.

[26] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems*, April 2000.

[27] S. Wu and B. Kemme. Postgres-R(SI):combining replica control with concurrency control based on snapshot isolation. In *Proceedings of the IEEE International Conference on Data Engineering*, 2005.