Middleware-based Data Replication: Some History and Future Trends

J.E. Armendáriz-Íñigo[1], Hendrik Decker[2], J.R. González de Mendívil[1] and F.D. Muñoz-Escoí[2]
[2] Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
Campus de Vera, 46022 Valencia, Spain
{*hendrik, fmunyoz*}*@iti.es*
[1] Dpto. Matemática e Informática
Universidad Pública de Navarra
Campus de Arrosadía, 31006 Pamplona, Spain
{*enrique.armendariz, mendivil*}*@unavarra.es*

Abstract

Middleware-based data replication is a way to increase availability, fault tolerance and performance of networked information systems without modifying the underlying DBMS core code. However, if such middleware is not properly conceived, it will introduce an overhead leading to poor response times. In this paper we study the progression of solutions for problems typically entailed by replication, which led to the use of middleware for avoiding inconveniencies associated with earlier approaches. We survey several existing and projected architectures –designed by our research groups– in terms of availability, consistency and fault tolerance.

1 Introduction

Data replication serves to increase the availability and fault tolerance of networked information systems while protecting against downtimes due to site failures. Transaction workload can be distributed among several interconnected servers in a well-balanced manner such that, whenever a site fails, its current workload can be seamlessly redirected to a network node that is still alive. Redirection of work may either be executed according to a predetermined network pattern or be decided upon dynamically, depending on various runtime parameters. However, the consistency of replicated data must be taken care of, i.e., updates must be propagated in a timely manner to all nodes. The exact meaning of "timeliness" may strongly depend on the application, user profile, network load, etc.

In this paper we study the progression of our replication middleware projects from the viewpoints of users, applications, protocols and databases. For users and applications, replication must be transparent, i.e., neither users nor applications need to change their behavior when moving from a centralized to a distributed architecture, since their system interface is supposed to remain the same, while availability, performance and fault tolerance are supposed to improve.

With regard to the protocols for replication, messaging and recovery, their implementation tends to be intertwined with tasks for transaction processing, concurrency handling and component coordination, which may easily drag down their efficiency. Hence, we are going to keep an eye on techniques for implementing protocols that allow them to focus on optimizing their own tasks without being unduly distracted by issues that are in fact alien to them. Moreover, the adequateness of choosing and adapting protocols and replication strategies to the diverse needs of different users, applications and network parameters is of our concern as well.

With regard to databases, it will be interesting to see to which extent built-in DBMS functionality can be used for supporting replication data management, and to which extent this may inhibit a targeted independence from the particularities of underlying database systems, or introduce undesirable redundancies beyond the multiplicity of copied data induced by replication.

The remainder is organized as follows. Section 2 surveys solutions for realizing replication that historically precede the middleware approach. Section 3 recapitulates the pros and cons of the main targets and results of the COPLA middleware, the main characteristics of which, beyond replication, is its object orientation. Section 4 reports on improvements brought about by the MADIS project, which abandoned object orientation in favor of standard JDBC interfacing to SQL databases. Section 5 outlines further advances of replication middleware with regard to higher availability, efficiency and adaptability, as envisaged in an upcoming project. Section 6 concludes.

2 Historical Background

Traditionally, database replication has been achieved by modifying the core code of the database management system (DBMS hereafter), as in [5, 6, 12, 15, 16, 17, 24]. Essentially, their modifications consist in maintaining an efficient access to the transaction log of modifications and adding a group communication support module, such as in Postgres-R [15] and Postgres-R(SI) [24]. This approach sports good

performance, since only a minimal overhead is introduced. However, it is not easily portable, hardly interoperable in networks of systems based on the products of different vendors, and difficult to maintain even for releases of new versions from the same brand.

In [5], all operations were propagated to all sites, which meant to run a considerable danger of distributed deadlocks. A natural improvement of this way of implementing consistent replication is the ROWAA (Read One Write All Available) approach, where operations are firstly performed exclusively on a single node and then, only the updates are propagated; no additional communication messages need to be transmitted over the network. This is realized, e.g., in [6], with Optimistic 2PL, a new kind of two-phase lock mechanism which distinguishes between local and remote transactions in order to anticipate and avoid possible deadlock situations. Nevertheless, as Optimistic 2PL propagates updates without any kind of ordering, this approach is still quite likely to run into distributed deadlocks.

To overcome this, some effort was invested in developing Group Communication Systems (GCS) [11, 8] that provide certain message ordering guarantees as well as mechanisms for detecting potentially crashed sites. The strongest delivery guarantee is the total order multicast, where messages are delivered to all nodes in the same order, thus avoiding distributed deadlocks altogether. This idea was adopted by the Postgres-R project [15, 16]. However, imposing a total order to all transactions is not necessarily the most effective way to avoid deadlocks, since a total order delivery may require several communication rounds [9], and indeed, non-conflicting transactions do not need to be delivered in total order.

Several improvements have been investigated to overcome the latency of total order delivery approaches. For instance, with optimistic atomic broadcast as described in [23, 17], messages are delivered as they are received, making possible a fast remote writeset application, although waiting for the final ordered message delivery in order to commit the transaction. Thus, only those remote transactions whose writeset did not follow the total order are rolled back, reapplying them in the correct order. In the generic broadcast presented in [1, 21], only the messages which belong to a conflicting class must be delivered in total order while the rest may be delivered without any ordering. Moreover, other communication techniques have been used, e.g. the epidemic algorithms discused in [12].

The strongest correctness criterion for database replication is the 1-Copy-Serializability property [5]. It provides for a transparent serial execution of a transaction over a single virtual data unit although there are possibly many physical copies that are distributed over different sites. This correctness property is adequate when the underlying DBMS provides serializable histories, but most databases on the market only provide Snapshot Isolation (SI) [4]. Therefore, new correctness criteria have been introduced, such as Generalized Snapshot Isolation (GSI) [10] and 1-Copy-SI (1CSI) [18] that define what can be considered as an acceptable SI level in a replicated environment.

As opposed to DBMS core code modification, a fundamentally different approach to implement replication is to deploy a middleware as an intermediate layer between clients and the DBMS, so that DBMS idiosynchrasies are hidden from both users and applications [18, 7, 14, 13, 20]. The major purpose of the middleware is to cater for replication consistency. For that, it has turned out to be helpful to extend the original underlying database schema definition using standard SQL features such as table creation, triggers, stored procedures, etc. [13] in order to maintain metadata needed for replication. Such extensions are supported by the middleware and can be derived automatically, i.e. without intervention from schema designers or users. Another approach, described in [20], uses conflict classes to define transactions issued by users, e.g. purchasing a book. For each purchase order, the system exactly knows the items to be bought, and the client who performs the update. Hence, record-level conflict classes can be implemented and each one of them has a master site. Transaction execution for all algorithms is roughly as follows. The client submits a transaction t to any site, as the replication system is transparent to him. This site immediately forwards the message to all sites. All sites append t to the queues of the basic conflict classes t accesses. Only the master executes the transaction whenever it is the first in all queues. It executes both read and write operations on the local database. Then, it multicasts the updates performed by t to the remote sites. Remote sites only apply these updates instead of reexecuting the entire transaction. In order for the middleware to send and apply writesets, it is assumed that the underlying DBMS provides two services: one to obtain the writeset of a transaction and another one to apply the writeset (for instance, in PostgreSQL it is the $clog$). Besides, these DBMS's facilities are used also in [18].

Thus, the middleware approach does not need to modify any piece of DBMS core code. The mentioned schema extensions enable a delegation of data management work to built-in DBMS functionality, so that the implementation of protocols can focus on their main purpose of replica management, without being overburdened by concurrency and transaction management. This has been described in more detail in [13]. In summary, the middleware approach sketched above introduces some overhead of creating and maintaining additional meta data tables, which amounts to a certain performance penalty, but it permits to get rid of dependencies on underlying DBMS peculiarities.

3 Database Replication, Truly Object-Oriented

The middleware architecture COPLA was developed around the turn of last century (cf. [14]). It was fully object-oriented: regardless of the underlying database, users and applications had a unique, transparent object-oriented view of, and access to, the stored data, and object state persistence was provided by the transaction and concurrency handling modules. COPLA provides several options of protocols (cf., e.g., [19, 22]), each of which trades off the conflicting goals of high availability and replication consistency in a different manner. Hence, each installation of COPLA for a given application could be configured according to the specific availability and consistency needs of that application, by opting for an appropriate protocol.

Unfortunately, the replication protocols had to re-implement several standard transaction and concurrency management features, such as lock management or a version system. A further development penalty was that applications had to be (re-)implemented on top of COPLA using the proprietary object definition language GODL [2]. Objects were fetched in the context of a transaction by way of a proprietary object query language GOQL [2]. Another unfortunate consequence of object orientation was that, for an underlying relational database, an object-relational mapping had to be accomplished, which entailed a significant performance penalty.

Client applications could access stored data by way of transactions using three different levels of consistency: *plain*, no consistency at all; *checkout*, with guarantees similar to the CVS tool for source-code management; and, *serializable*, with the same meaning than the corresponding transaction isolation level. One of the contributions of this project was that some of its replication protocols [19] were able to manage all these consistency levels simultaneously. Thus, no additional support was required for dealing with concurrent transactions that use different consistency levels, either initiated by one or many applications. None of the recent publications in this area has suggested a solution of this kind, and it simplifies a lot the effort needed for concurrency management when multiple isolation or consistency levels must be used. Without this, if an application uses subsets of transactions that need different consistency (or isolation) levels, it is compelled to use the stricter level in order to guarantee correct execution results, and this may lead to some abortions of those transactions that were designed to use a relaxed level, and that would have committed if it had been available.

Finally, the COPLA system was also used as a testbed for developing total-order broadcast protocols with uniform and optimistic delivery in WAN environments. The replication protocols described in [22] use a broadcast of this kind [23] and provided a good basis for replica management in WANs.

4 A JDBC-Based Middleware

The middleware MADIS [13] is a follow-up to COPLA. Instead of object orientation, it provides a straightforward JDBC interface, so that applications do not need to be adapted or re-implemented. Replica consistency control is implemented very concisely by a schema extension which exclusively uses built-in functionality of the underlying DBMS. All of this reduces the penalties of development and performance mentioned in Section 3 significantly. On the other hand, the overhead introduced by creating and managing schema extensions for administering meta data for replication management, concurrency control and failure recovery turns out to be reasonably small. A performance comparison of COPLA and MADIS has been already presented in [13] and it showed that MADIS is between 4 and 20 times faster than COPLA, depending on the load and operations being considered.

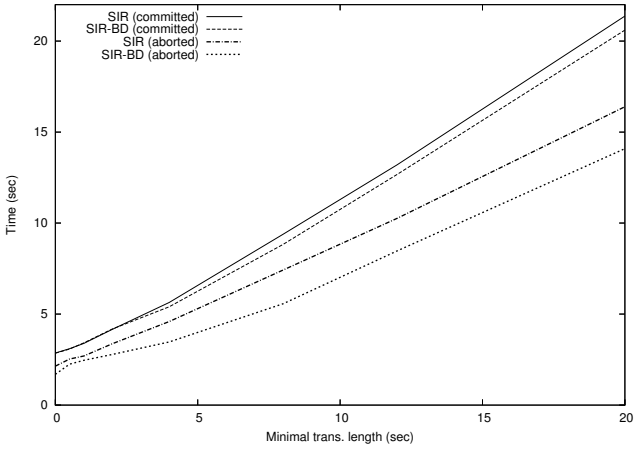

Figure 1. Automated conflict detection improvements

Another contribution of this middleware is the support for detecting conflicts among transactions that has been implemented in its plugs into the DBMS. To this end we use a thread that periodically scans the system catalog table that provides information on transactions currently blocked due to the lock management. This system catalog table is present in multiple DBMSs such as PostgreSQL, MySQL, Microsoft SQL Server, etc. Our mechanism allows the assignment of priorities to transactions, and automatically aborts a locker transaction if it has lower priority than the

blocked one. Since many replication protocols based on total order broadcasts abort local transactions if they collide with remote writesets when they have to be applied, this technique might simplify the transaction management at the middleware level. In [3] we have thoroughly described this technique, providing some measurements that show its performance improvements.

As an example, Fig. 1 shows how the mean transaction completion time has been improved when our conflict detection technique has been applied to the SI-Rep protocol described in [18], with a load that provided a 5% of writeset collisions among concurrent transactions. To this end, we show two implementations of this protocol. The first one, named SIR in the figure, uses an optimized conflict detection at the middleware layer that only needs to compare two integer values per writeset (the value of an integer primary key and the length of the writeset), and the second one (SIR-BD) uses our technique. As shown in the figure, our approach reduces the response time of those transactions that finally must be aborted (since it is able to abort them as soon as possible), and this even provides an improvement on the response time of committed transactions since the overall load is also reduced.

5 Future Trends in Database Replication Middlewares

Future developments to follow up on MADIS are envisaged to make replication support more dynamic. Dynamic support may evolve along two orthogonal but complementary dimensions. One is to support replication not only for static networks with fixed numbers of nodes, but also for more dynamic networks, where the network topology is not necessarily known in advance, or may vary from session to session, and also during a session. The other is to enable a higher flexibility of choosing, plugging in and swapping protocols in order to optimally adapt the replication management and recovery strategies to the different installations, but also different concurrent applications and users with different profiles, or different situations as determined by the number of alive and broken nodes, the current network traffic, data channel capacity, transmission rates, device properties, billing modalities, etc.

From the point of view of network designers and users, network dynamics may behave *intentionally* or *unintentionally*. Intentional changes may be *scheduled* (e.g., opening and closing hours of businesses, time intervals of electronic conferences) or *ad-hoc* (e.g., casual user log-ons and log-offs). An intermediate and often hybrid kind of partly scheduled, partly ad-hoc dynamics in information system networks needs to be catered for in P2P and Grid networks, due to the strong autonomy of network nodes which may enter and leave sessions in a more or less planned manner.

Unintentional changes may be *accidental* (e.g., crashes of sites or links due to software or hardware failures) or *malicious* (e.g., crashes due to security breaches). Clearly, the handling of intentional changes can be expected to be much easier than unintentional ones, since the effects and consequences of the former can be anticipated much better. On the other hand, it generally does not matter much, from the point of view of maintaining high availability and consistency, whether unintentional events which change the network are caused accidentally or maliciously, since the moment in which they occur cannot be predicted with sufficient precision. This distinction only makes sense for security protocols, which however are out of the scope of this paper. Yet, making a difference between "common" and "byzantine" failures (both of unintentional kind, but skew to distinguishing accidental and malicious failures) can be helpful, mainly for recovery protocols.

For each case of network dynamics as classified above, specific protocols for replication consistency and error recovery are planned to be developed in the upcoming project CONFIA. The idea is to configure and reconfigure the middleware such that the often conflicting requirements of availability, timeliness, performance and consistency can be optimized, by choosing, plugging in and swapping dynamically those protocols that fit best the dynamically changing requirements of different users, applications and network properties.

Usually, at most just a single manner of replication, a fixed consistency maintenance scheme and a uniform policy for availability and failover management is supported in networked information systems. A more flexibly adaptable replication architecture with a sufficiently large repertoire of different protocols is therefore desirable, which is the goal of CONFIA. Several such protocols, although within a narrower scope of diversity, have already been developed for the MADIS and COPLA platforms. COPLA was capable of providing support for different strategies and protocols of replication and fault tolerance, and was configurable for plugging in different protocols, but only off-line. MADIS went much further, by providing on-line support for protocol swapping, but not for simultaneously running applications nor for different needs and profiles of collaborating or concurrent users or programmed agents. This flexibility will be achieved by a simultaneous maintenance of metadata for all repertoire protocols, such that each of them can be plugged in seamlessly at any moment while the network is in operation.

6 Conclusion

We have studied a progression of replication architectures for enhancing the availability, fault tolerance and performance of network-based information systems. We have

done so from the viewpoints of users, applications, protocols and databases. We have noticed that each different viewpoint corresponds to a dimension of its own by which the design, choice and dynamic exchange of adequate replication strategies may be guided.

Acknowledgments

This work has been supported by the Spanish Government research grant TIC2003-09420-C02.

References

[1] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Thrifty generic broadcast. In M. Herlihy, editor, *DISC*, volume 1914 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 2000.

[2] J. E. Armendáriz, J. J. Astrain, A. Córdoba, and J. Villadangos. Implementation of an object oriented query language compiler for replicated architectures. In E. Pimentel, N. R. Brisaboa, and J. Gómez, editors, *JISBD*, pages 441–450, 2003.

[3] J. E. Armendáriz, F. D. Muñoz, M. I. Ruiz, J. R. G. de Mendívil, and L. Irún. Dealing with Writeset Conflicts in Database Replication Middlewares. Technical Report ITI-ITE-05/11, ITI, 2005.

[4] H. Berenson, P. A. Bernstein, J. Gray, J. Melton, E. J. O'Neil, and P. E. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD Conference*, pages 1–10. ACM Press, 1995.

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

[6] M. J. Carey and M. Livny. Conflict detection tradeoffs for replicated data. *ACM Trans. Database Syst.*, 16(4):703–746, 1991.

[7] E. Cecchet, J. Marguerite, and W. Zwaenepoel. C-jdbc: Flexible database clustering middleware. In *USENIX Annual Technical Conference, FREENIX Track*, pages 9–18. USENIX, 2004.

[8] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

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

[10] S. Elnikety, W. Zwaenepoel, and F. Pedone. Database replication using generalized snapshot isolation. In *SRDS*, pages 73–84. IEEE Computer Society, 2005.

[11] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Dep. of Computer Science, Cornell University, 1994.

[12] J. Holliday, R. C. Steinke, D. Agrawal, and A. E. Abbadi. Epidemic algorithms for replicated databases. *IEEE Trans. Knowl. Data Eng.*, 15(5):1218–1238, 2003.

[13] L. Irún, H. Decker, R. de Juan, F. Castro, J. E. Armendáriz, and F. D. Muñoz-Escoí. MADIS: A slim middleware for database replication. In *Euro-Par*, volume 3648 of *LNCS*, pages 349–359. Springer, 2005.

[14] L. Irún, F. Muñoz, H. Decker, and J. M. Bernabéu-Aubán. COPLA: A platform for eager and lazy replication in networked databases. In *5th Int. Conf. Enterprise Information Systems (ICEIS'03)*, volume 1, pages 273–278, Apr. 2003.

[15] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In *VLDB*, pages 134–143, 2000.

[16] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[17] B. Kemme, F. Pedone, G. Alonso, A. Schiper, and M. Wiesmann. Using optimistic atomic broadcast in transaction processing systems. *IEEE Trans. Knowl. Data Eng.*, 15(4):1018–1032, 2003.

[18] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *SIGMOD Conference*, 2005.

[19] F. Muñoz-Escoí, L. Irún-Briz, P. Galdámez, J. Bernabéu-Aubán, J. Bataller, and M. Bañuls. GlobData: Consistency protocols for replicated databases. In *YUFORIC'2001*, pages 97–104. IEEE-CS, 2001.

[20] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. MIDDLE-R: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4):375–423, 2005.

[21] F. Pedone and A. Schiper. Generic broadcast. In P. Jayanti, editor, *DISC*, volume 1693 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 1999.

[22] L. Rodrigues, H. Miranda, R. Almeida, J. Martins, and P. Vicente. The globdata fault-tolerant replicated distributed object database. In *EurAsia-ICT*, volume 2510 of *LNCS*, pages 426–433. Springer, 2002.

[23] P. Vicente and L. Rodrigues. An indulgent uniform total order algorithm with optimistic delivery. In *SRDS*, pages 92–101. IEEE Computer Society, 2002.

[24] S. Wu and B. Kemme. Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation. In *ICDE*, pages 422–433. IEEE-CS, 2005.