Implementing Database Replication based on Group Communication

Bettina Kemme
School of Computer Science
McGill University, Montreal, Canada
kemme@cs.mcgill.ca

Abstract

Many proposals have been made to exploit the rich semantics of group communication systems to support database replication. So far, these proposals have either used simulation studies or simple prototype implementations. This paper analyzes what is still missing and what has to be addressed – both from a theoretical and a practical perspective – in order to build "real" systems.

For many years, developers of group communication systems have claimed that database replication is an interesting application for their multicast primitives. Over the last years, several research groups have had a closer look into this claim and started to analyze how group communication primitives can support replica control in database systems, for instance [10, 19, 18, 15, 16, 14, 21, 13, 4, 2]. The main ideas of these proposals are to use the ordering guarantees of multicast primitives to serialize conflicting transactions, and to simplify atomic commit protocols by using reliable or uniform reliable multicast. Most of the results are published in distributed systems conferences, and so far, they have received little attention in database conferences. There, the focus has been on quorum protocols [22] (for fault-tolerance) and adaptive lazy replication schemes (for performance and scalability) [9, 6, 5]. Out of this wide range of research directions, however, only few ideas have found their way into commercial database systems. Commercial solutions are either highly specialized backup systems, or they choose performance over consistency. Practitioners recommend to not use these replication strategies for systems with high update rates: "...In my experience most of these (replication) schemes end up being more trouble than the benefit they bring" [7].

Looking at this low transfer rate between research results and commercial systems, I believe that the use of group communication systems for database replication will only then become reality, if the theoretical results proposed so far will be validated through the development and evaluation of prototypes and small "real" systems (e.g., in the form of open-source software modules). While simulation results of the proposed approaches have been very promising (e.g., [19, 16, 13]), only prototypes will be able to convince industry of the usefulness of the approach. Such research is especially challenging:

- It requires in-depth knowledge in both database systems and distributed systems, and solid background in both theory and systems.
- Two different systems are combined (database and group communication systems). Hence, there exist many alternatives for the overall architecture showing different degrees of interleaving and interaction.
- Each realistic solution has to consider all issues related to database replication: concurrent execution of transactions, failure-handling, recovery, full support of the SQL standard, and user transparency. Each of these issues by itself is already challenging, combining them make a "real" implementation very hard.

There exist two alternatives of approaching the problem: an independent middleware based replication tool, or the integration of the replication semantics into the database system. In the first case, the replication tool could be an extension to, e.g. CORBA or J2EE. In the second case, the approach would follow the lines of database internal replication schemes as implemented in Oracle, Sybase etc. I am aware of three prototype implementations. Postgres-R [15] integrates replication into the kernel of the database system PostgreSQL [20] and can use both Ensemble [11] and Spread [3] as group communication system. Amir et al. [2] describe a middleware based replication tool using Spread and PostgreSQL. The PostgreSQL

development team is interested in incooperating the ideas of these two prototypes into their distribution. Jiménez-Peris et al. [14] also developed a middleware system, this time based on Ensemble and PostgreSQL.

These prototypes already handle a considerable amount of issues but still lack functionality that is crucial in "real life" systems. In order to add this functionality, both more theoretical work and smart strategies for translating theoretical solutions into practical implementations are needed:

- *Concurrency Control*: Although following the total order in case of conflicts, it should be possible to execute transactions concurrently whenever they do not conflict. If replication is implemented as middleware, the middleware must provide such a concurrency control mechanism. Since it receives SQL or other high-level statements from the client it is hard to determine which operations actually conflict. As a consequence, Amir et al. [2] execute update transactions serially, Jiménez-Peris et al. [14] lock entire relations or partitions, which only allows for restricted parallelism. If replication is performed within the database system, things seem to be easier. Many of the more theoretical research papers have proposed various adjustments to existing concurrency control protocols in order to handle replication [13, 19]. This goes so far as to exploit weaker message ordering mechanisms (like FIFO or causal) in order to provide more concurrency, e.g. [21]. However, it is very hard to transform these proposals into practical solutions since the concurrency control module usually heavily depends and interacts with the other components of the database system (e.g., storage manager, access paths, buffer manager, etc). Hence, concurrency control cannot be simply added as an independent component to the database kernel but will probably be a specialized solution for a specific database system. Postgres-R [15] provides a restricted form of fine-granularity concurrency control, and discussions with the PostgreSQL development team have shown the willingness to include a (rather simple) replication based concurrency control into the PostgreSQL engine as a long-term goal.
- *Transaction Model:* Most of the transaction models proposed so far have some restrictions. Amir et al. [2] assume that a transaction consists of a single SQL statement or is a single stored procedure executed within the database system. However, transactions often consist of more than one operation and these operations must be executed one by one since operations might depend on each other. Jiménez-Peris et al. [14] and Postgres-R [15] overcome this problem by executing a transaction first locally and multicasting all changes in a single message at the end of the transaction. Keeping track of all changes performed by a transaction might lead to a considerable overhead if the database system does not offer adequate support. Other approaches delay write operations, include read operations into the messages, or send messages for each operation [19, 1, 13] – each of these mechanisms having its own advantages and disadvantages. In my opinion, middleware solutions might have to live with compromises. In contrast, implementations within the database system have more flexibility in solving these issues.
- *Failure and Recovery*: Talking to people from industry [8], it has become clear that failure and recovery are very crucial issues. Most importantly, the reconfiguration process should be as automatic as possible, and should not interrupt ongoing transaction processing significantly. Furthermore, no replica may miss any transactions. Since the recovery process can take place in the middleware or with support of the database, there are many interesting alternatives. Postgres-R currently copies the entire database on recovery (still a suboptimal solution). Amir et al. [2] log all messages in the middleware layer. If sites are down for a short down, a restarting site receives all missed messages, otherwise a state transfer might take place. [12, 17] have suggested more flexible schemes providing an optimized state transfer that does not interrupt transaction processing in the system. In general, reconfiguration is an issue that will need much further analysis. Both the database system and the group communication system provide plenty of functionality for failure handling and recovery. We are currently working at a clear interface between these two systems such that the functionality of both components can be exploited: the work should be distributed and not duplicated. For instance, the database system performs extensive logging and one should take advantage of it in order to avoid logging of messages in the middleware layer.
- *Transparency*: Transparency is a crucial issue. The main question is of how to embed middleware based replication tools. They might be part of an existing middleware infrastructure (e.g., CORBA or J2EE) and applications are developed using this middleware. Alternatively, the replication tool is put between legacy application and database system without the application being aware of it [2].
- *Further Issues:* Within here, I only want to mention some further interesting developments:
  When looking at wide area replication priorities might shift: On the one hand, weaker and flexible

transaction models might be acceptable and concurrency within a single site might not be crucial. On the other hand, adaptive failure-handling and recovery are even more important. Message latency will become a significant factor, and optimistic strategies will have to be considered.

When looking at completely heterogeneous environments or systems in which access is restricted (e.g., access is only available through web-based interfaces), reasonable models have to be developed that are able to handle the given restrictions and still provide acceptable guarantees.

As a summary, I believe that database systems can take advantage of group communication systems and provide better replication mechanisms as currently provided in commercial systems. However, I believe that the research part has not yet been completed. One of the main tasks right now is to prove that the solutions can be transferred into real systems. Furthermore, the theoretical framework must be extended and adjusted in order to properly address the requirements of a real environment.

References

[1] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi. Exploiting Atomic Broadcast in Replicated Databases. In *Euro-Par Conf.*, 1997.

[2] Y. Amir, C. Danilov, M. Miskin-Amir, J. Stanton, and C. Tutu. Practical Wide Area Database Replication. Technical Report CNDS-2002-1, Johns Hopkins University, 2002.

[3] Y. Amir and J. Stanton. The spread wide area group communication system. Technical report, Department of Computer Science, The John Hopkins University, CNDS-98-4, 1998.

[4] Y. Amir and C. Tutu. From Total Order to Database Replication. In *Proc. of Int. Conf. on Distr. Comp. Systems (ICDCS)*, 2002.

[5] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, and A. Silberschatz. Update propagation protocols for replicated databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1999.

[6] P. Chundi, D. J. Rosenkrantz, and S. S. Ravi. Deferred updates and data placement in distributed databases. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 469–476, New Orleans, Louisiana, February 1996.

[7] S. Allamaraju et. al. *Professional Java Server Programming, J2EE Edition.* Wrox Press, 2000.

[8] GBorg. *PostgreSQL related Projects.* http://gborg.postgresql.org/project/pgreplication/projdisplay.php, 2002.

[9] J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1996.

[10] R. Guerraoui. Revisiting the relationship between non-blocking atomic commitment and consensus. In *Proc. of the IEEE Int. Workshop on Workshop on Distributed Algorithms (WDAG-9)*, 1995.

[11] M. Hayden. The Ensemble System. Technical Report TR-98-1662, CS Dept. Cornell Univ., Jan. 1998.

[12] J. Holliday. Replicated database recovery using multicast communications. In *Proc. of the IEEE Symposium on Network Computing and Applications (NCA2001)*, 2001.

[13] J. Holliday, D. Agrawal, and A. El Abbadi. The Performance of Database Replication with Group Communication. In *Proc. of Int. Symp. on Fault-Tolerant Computing (FTCS)*, 1999.

[14] R. Jiménez-Peris, M. Patiño-Martínez, B. Kemme, and G. Alonso. Improving the Scalability of Fault-Tolerant Database Clusters. In *Proc. of Int. Conf. on Distr. Comp. Systems (ICDCS)*, 2002.

[15] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, A new way to implement Database Replication. In *Proc. of Int. Conf. of Very Large Databases (VLDB)*, 2000.

[16] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Transactions on Database Systems*, 25(3), 2000.

[17] B. Kemme, A. Bartoli, and O. Babaoglu. Online Reconfiguration in Replicated Databases Based on Group Communication. In *Proc. of the Int. Conf. on Dependable Systems and Networks (DSN)*, 2001.

[18] F. Pedone and S. Frolund. Pronto: A Fast Failover Protocol for Off-the-shelf Commercial Databases. In *Proc. of the Int. Symp. on Reliable Distributed Systems (SRDS)*, 2000.

[19] F. Pedone, R. Guerraoui, and A. Schiper. Exploiting Atomic Broadcast in Replicated Databases. In *Proc. of Euro-Par*, 1998.

[20] PostgreSQL. *v6.4.2.* http://www.postgresql.com, January 1998.

[21] I. Stanoi, D. Agrawal, and A. El Abbadi. Using broadcast primitives in replicated databases. In *Proc. of the Int. Conf. on Distributed Computing Systems (ICDCS)*, 1998.

[22] A. Wool. Quorum systems in replicated databases: Science or fiction? *Bulletin of the Techn. Committee on Data Engineering*, 21(4), 1998.