Evaluating Database Replication Protocols in the MADIS Middleware Architecture

J.E. Armendáriz*, J.R. Juárez*, J.R. Garitagoitia*, J.R. González de Mendívil* and F.D. Muñoz-Escoí†
*Dpto. de Matemática e Informática, Universidad Pública de Navarra, 31006 Pamplona, Spain
Email: {enrique.armendariz, jr.juarez, joserra, mendivil}@unavarra.es
†Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, 46022 Valencia, Spain
Email: fmunyoz@iti.upv.es

I. Introduction

Optimistic 2PL (O2PL)[1] was one of the first concurrency control algorithms specially designed for replicated databases and not following the replication policy established in [2]. O2PL showed several advantages when compared with other general concurrency control approaches (such as distributed 2PL, basic timestamp ordering, wound-wait, or distributed certification): (a) As many of them, it does not need to propagate readsets in order to detect concurrency conflicts. Read locks are only locally managed, using the support provided by the underlying Database Management System (DBMS). (b) It only needs constant interaction [3], delaying all remote write-lock requests until commit time, being thus an optimistic variation of the distributed 2PL approach [4]. This ensures a faster transaction completion time than those protocols based on linear interaction. (c) Its use of locks, although optimistic, guarantees a lower abortion rate than that of the timestamp-based approaches [1]. The principles of O2PL have been used in many modern database replication protocols [5] based on total order broadcast [6], removing thus the need of using the Two Phase Commit (2PC) protocol in order to terminate the transactions, and improving in this way the protocol outlined in [1]. We propose two new replication protocols directly based on the ideas discussed above, and implemented in a middleware called MADIS [7].

A middleware-based implementation has to necessarily add some collection and management tasks that reduce the performance of the resulting system. On the other hand, the resulting system will be easily portable to other DBMSs. In both protocols we have eliminated the need of lock management at the middleware layer. For this purpose, we rely on the local concurrency control, adding some triggers that will be raised each time a transaction is blocked due to a lock request. Each time a client issues a transaction (*local transaction*), all its operations (i.e. all reads and writes) are locally performed on a single node called the *transaction master site*. The remainder sites enter in the context of this transaction when the user wants to commit. All write operations are grouped and sent to the rest of available sites, at this moment is when the two protocols differ, since the former uses the basic service and the latter employs a total order. Updates are applied in the rest of sites in the context of another local transaction (*remote transaction*) on the given local database where the message is delivered.

The first replication proposed is called BULLY. It is based in a 2PC [4] atomic commitment protocol such as O2PL [1]; it needs only a uniform reliable broadcast and a globally unique priority value associated to each transaction. BULLY requires two communication phases in order to commit a transaction, since all its operations are firstly executed at a given site. The first phase comprises: update propagation, priority check among current active transactions at the rest of sites (to avoid distributed deadlocks), and applying the updates on the local DBMS. This algorithm is called BULLY due to the fact that updates performed at remote nodes rollback all conflictive (neither committed nor aborted) transactions whose priority is lower than its own. Once this process is finished at each remainder site, it sends a message saying it is ready to commit. The second phase starts when all sites are ready to commit. The master site multicasts a commit message to the rest of sites (the same may be applied for an aborted transaction). Therefore, it is also able to manage unilateral aborts. The second replication protocol, called Total Order Replication Protocol with Enhancements (TORPE), uses the total order multicast primitive [6] to order conflicting transactions instead of the BULLY priority function, in a similar way as in [5]. TORPE replaces the first communication phase of the previous one with one total order broadcast [6]. If the site where the transaction is executed notices this message as the first one delivered, it directly multicasts a commit message, otherwise, it multicasts an abort message. However, it is not able to manage unilateral aborts.

II. Experimental results on MADIS

We are currently ending the implementation of MADIS while implementing our replication protocols on it [7]. The results presented here are preliminary ones and merely point out the comparison between these protocols in a middleware architecture. These results have been performed in two different environments, since the first one does not have enough workstations to execute the second test. We firstly use a cluster of 4 workstations with full duplex Gigabit Ethernet (Mandrake 10.0, Pentium III 800MHz, 768MB main memory, 40GB SCSI disk) and secondly a cluster of 8 workstations with full duplex Fast Ethernet (Fedora Core 1, Pentium IV 2.8GHz,

1GB main memory, 80GB IDE disk). In all our tests, we use PostgreSQL 7.4 as the underlying DBMS. Spread 3.17.3 is in charge of the group communication system for both protocols. The database is composed by 25 tables with 100 records each one. The experimental results consist of executing non-conflicting transactions composed of a number of update operations varying the number of clients.

In the first experiment, transactions averaging 4 updates are executed by a range of clients supporting different workloads using both replication protocols. Figure 1 shows the results for the BULLY and the TORPE protocols respectively. Results obtained in these figures determine the performance of both protocols with the same load of transactions. As shown in the figures, TORPE behaves better than BULLY, due to the fact that BULLY has to wait for the application of updates at all nodes and the reception of the respective ready messages. TORPE has only to wait for the total order delivery of the remote message and is not affected by the overhead introduced by remote transactions.

The second experiment is directly related to the scalability of the replication protocols [2]. We perform a proof varying the number of sites 2, 4 and 8 respectively. The number of clients are distributed throughout the nodes ranging from 1 to 16 and the load introduced into the system remains constant to 8 TPS. We performed 500 transactions averaging 4 update operations each per client. Results introduced in Figure 2 show that TORPE behaves fine for a few number of clients and nodes but its results are comparable to those obtained by BULLY with a higher number of clients and sites. This is due to the fact that the latency of total order delivery grows with the number of nodes, as it takes more time to agree on the delivery order. BULLY grows linearly with the number of nodes and TORPE grows much faster.

ACKNOWLEDGMENTS

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

REFERENCES

[1] M.J. Carey and M. Livny, "Conflict detection tradeoffs for replicated data.," *ACM Trans. Database Syst.*, vol. 16, no. 4, pp. 703–746, 1991.

[2] J. Gray, P. Helland, P.E. O'Neil, and D. Shasha, "The dangers of replication and a solution.," in *SIGMOD Conference*, H. V. Jagadish and Inderpal Singh Mumick, Eds. 1996, pp. 173–182, ACM Press.

[3] Fernando Pedone, Matthias Wiesmann, André Schiper, Bettina Kemme, and Gustavo Alonso, "Understanding replication in databases and distributed systems.," in *ICDCS*, 2000, pp. 464–474.

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

[5] Bettina Kemme and Gustavo Alonso, "A new approach to developing and implementing eager database replication protocols.," *ACM Trans. Database Syst.*, vol. 25, no. 3, pp. 333–379, 2000.

[6] G. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: a comprehensive study.," *ACM Comput. Surv.*, vol. 33, no. 4, pp. 427–469, 2001.

[7] L. Irún-Briz, H. Decker, R. de Juan-Marín, F. Castro-Company, J.E. Armendáriz, and F.D. Muñoz-Escoí, "MADIS: A slim middleware for database replication.," in *Euro-Par*. 2005, Lecture Notes in Computer Science, Springer.

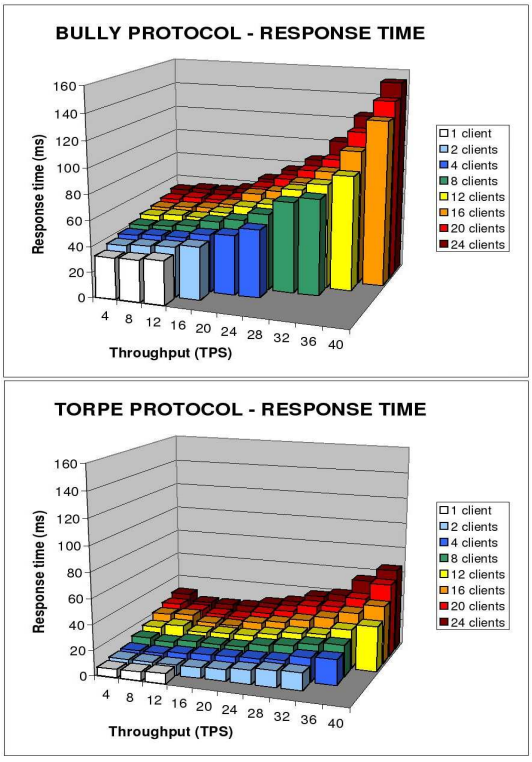

Fig. 1. Performance Analysis: Response time with four operations per transaction in a system with four sites.

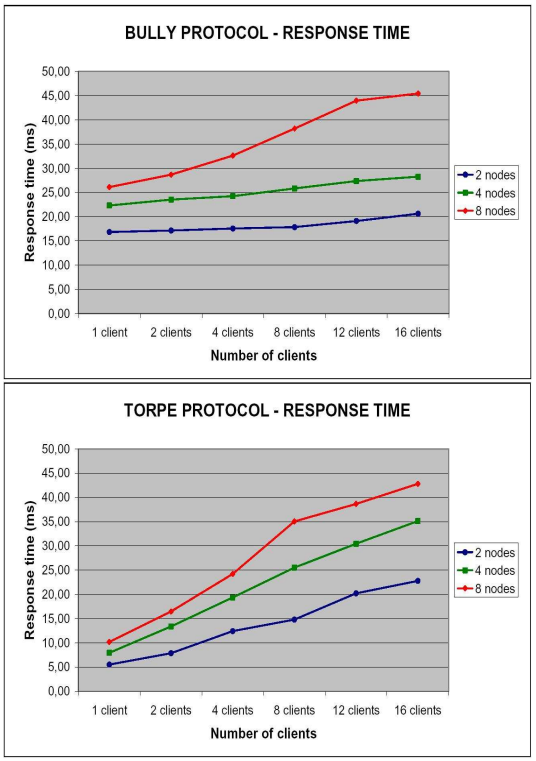

Fig. 2. Scalability Analysis: Response time with four operations per transaction.