Improving the Scalability of Fault-Tolerant Database Clusters*

R. Jiménez-Peris†, M. Patiño-Martínez†
School of Computer Science
Technical University of Madrid (UPM), Madrid, Spain
{rjimenez, mpatino}@fi.upm.es

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

G. Alonso
Department of Computer Science
Swiss Federal Institute of Technology (ETHZ), Zürich, Switzerland
alonso@inf.ethz.ch

1. The Problem

The increasingly pervasive use of clusters as the preferred platform for large information systems makes replication a central element of modern system architectures. Clusters offer the possibility to use off-the-shelf components to build more powerful information systems by simply adding more sites. This *scale out* approach (adding more elements) contrasts with the more traditional *scale up* approach (using more powerful elements) in that there are no fundamental limitations to how much a system can scale out. From the hardware point of view, a system can only scale up to a certain limit and the cost is exponential with the scale up factor. For a cluster system, in principle there are no limits to how many sites can be added. Unfortunately, things are not that straightforward. In a scale up approach, the software architecture does not play a big role: the better the hardware, the faster the application. In a scale out approach, if one is not careful with the software design, all the advantages can be quickly lost. Replication, for instance, is one of the software aspects that require particular attention since an improper design will turn a large cluster into a very reliable but very expensive version of a single machine.

This problem is particularly acute in information systems like those running web sites for electronic commerce, online auctions, or large data warehouses. Most of these systems are based on clusters where a number of servers simultaneously share the load and act as a backup to each other. This dual functionality is increasingly important as the system grows: additional sites can thereby increase both the processing capacity and the availability of the system. For this to work, data needs to be efficiently replicated across all servers (for availability) and the load partitioned so that each server can take care of part of it (for performance). The latter requirement holds in many systems. The limiting factor today is that existing data replication protocols are entirely inadequate for cluster environments.

Conventional, text book data replication protocols [1] have proven to be impractical for all intents and purposes: they do not scale and create huge overheads [2]. More modern approaches, like those that combine total order multicast with transactional properties, are for the most part theoretical constructs never implemented in a real system. The ones implemented, like Postgres-R [3], require to modify the underlying database, something that is not always feasible. Ideally, what is needed is both a protocol that can provide replication for both performance *and* availability as well as an implementation that is not intrusive.

2. A Solution

In this work we have implemented an eager scalable replication protocol [4] atop an existing database system. The applications we have in mind are clusters of data servers where requests can be divided into disjoint categories, a typical situation in e-commerce applications. The replication protocol we propose guarantees consistency at all times so that replicas can be used as backups for other replicas. It is also designed to minimize the processing overhead of replication, thereby allowing a degree of scalability that is much better than what is currently possible with existing commercial systems. The contribution of this work is to demonstrate that data replication can be implemented without severely limiting the scalability of the cluster and that this does not require to modify existing tools but can efficiently be done as an additional middleware layer. Performance and availability gains aside, this is one of the most attractive features of the system we propose.

The system we propose is a middleware layer situated

† This work has been partially supported by the Spanish National Research Council CICYT, grant #TIC98-1032

between clients and databases as shown in Fig. 1. Thus,

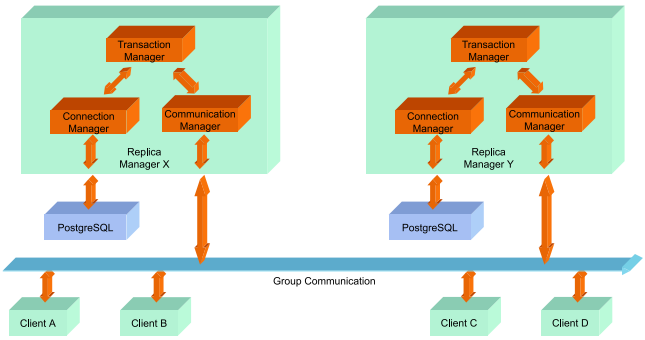

Figure 1. Middleware architecture

it is possible to obtain a replicated database out of non-replicated databases.

Two services are needed from the database in order to enable efficient replication: one to get the updates performed by a transaction and another one to apply the updates of a transaction. These services are usually implemented in commercial databases although they are not usually public. We have just added these two services to PostgreSQL to implement our prototype. For a set of homogenous databases, the updates can be in binary form (as it has been the case in our experiments), whilst for heterogeneous databases the updates can be represented with a string containing the corresponding update SQL statement. The advantage of just propagating the updates is that the work of parsing and executing the SQL statement is not repeated at every replica. This is especially interesting when the SQL statement scans a whole table to update a few tuples, and in general for transactions with a high-percentage of read operations with respect to write operations. This is why it is possible to scale despite using a read-one write-all approach.

3. Preliminary Results

3.1. Response Time with a Constant Load

The first question we addressed is whether the protocol we propose really solves the limitations of conventional replication protocols (e.g., those described in [1]). Gray et al. [2] showed that these conventional protocols do not scale and, in particular, that under a constant load increasing the number of replicas would increase the response time of update transactions and produce higher abort rates. To test the protocol, we have compared the scalability in terms of response time of our solution with that of a commercial product that implements replication based on standard distributed locking. The behavior of the distributed locking commercial solution did not scale at all: for a fixed load, the response time increased with the number of sites. This is due to the fact that distributed locking creates a significant amount of redundant work in the system, so much that adding extra sites becomes a liability more than an asset. Our system was quite stable. For a number of up to 20 sites, the response time did not vary, indicating that the system is not adding any extra latency when more sites are added.

3.2. Scale out

In a second experiment we measured the scale-out provided by the system for different system configurations of up to 15 sites, with different transaction profiles of 100%, 50%, and 0% updates. As can be seen in Fig. 2 the system presented linear scalability for read-only transactions, and provided a reasonable scale-out for a 50% updates. The system even scaled in the worst case of 100% updates. This scalability has been possible due to SQL statements are parsed and executed at a single site as well as read operations, whilst the rest of the sites just apply the resulting updates.

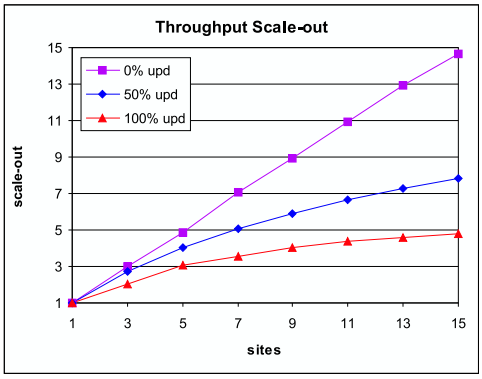

Figure 2. Scalability of the system for different transaction loads

References

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

[2] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proc. of the SIGMOD*, pages 173–182, Montreal, 1996.

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

[4] M. Patiño Martínez, R. Jiménez Peris, B. Kemme, and G. Alonso. Scalable Replication in Database Clusters. In *In Proc. of Int. Conf. on Distributed Computing, DISC'00. Toledo, Spain*, volume LNCS 1914, pages 315–329, Oct. 2000.