Enhancing the Availability of Networked Database Services by Replication and Consistency Maintenance *

Hendrik Decker, Francesc Muñoz, Luis Irún, Antonio Calero, Francisco Castro,
Javier Esparza, Jordi Bataller, Pablo Galdámez and Josep Bernabéu

Instituto Tecnológico de Informática, Universitat Politécnica de Valencia
{hendrik,fmunyoz,lirun,acalero,fcastro,jesparza,bataller,pgaldam,josep}@iti.upv.es

Abstract

We describe a middleware platform for maintaining the consistency of replicated data called COPla (Common Object Platform). The purpose of replication is to enhance the availability of data and services in distributed database networks. That is orthogonal to recovery strategies of backed-up snapshots, logs and other measures to alleviate database downtimes. A range of different consistency modes ensures the correctnes of replicated data.

1. Introduction

Availability of data and services is strategically critical for an increasing number of business processes. Traditionally, ensuring availability has meant to take measures for avoiding negative ramifications of database downtimes. That is, downtimes are tentatively absorbed by high investments in additional hardware, and attempted to be marginalized by sophisticated strategies of backups and recovery.

With the advent of the internet and its role as a backbone of networked databases, an orthogonal way of enhancing the availability of data has become feasible, viz. by replicating data in several network nodes. However, the drawback of replication is a potential overhead for maintaining the consistency of replicated data.

In this paper, we show how to reconcile the conflicting goals of enhancing the availability of services by replication of data, on one hand, and maintaining their consistency, on the other. We are going to describe the architecture of COPla [6], with an emphasis on its support for replication and consistency. Various protocols which can be plugged into COPla have been described in [11, 10, 7].Devoid of a closer look at consistency protocols, an overview of COPla has been given in [8, 12]. The first main contribution of this paper consists in a synopsis of features hitherto described only as isolated aspects, in the cited references. The second is the presentation of significantly improved performance results over those reported in [7].

In a network of distributed databases, service applications may start multiple database sessions. Each service or even each session may use different consistency modes, according to the particular needs of a given service. Such networks are usful for enterprises with several branch offices or chains of stores, such as banks, hypermarkets, etc. While some services may use on-site generated data in local branches, others may also use information generated in other branches and offices. However, the availability, timeliness and reliability of answers to queried data may be severely hampered and network traffic may be exasperatingly congested, since transmission channels, bandwidth, powerful harware etc may be a scarce resource. Particularly for web services in wide area or mobile networks of databases, the web may be very busy, and services may have to cope with network partitions caused by downtimes of local nodes, broken links to networked neighbors and the like.

To avoid such downsides, networked databases are in need of effective means to warrant a sufficient degree of availability. This paper discusses a platform conceived to enhance the availability of networked data and services. Somehow analogous to a flexible routing of data transmissions through networks where not all nodes are always up and running, our basic idea to ensure high availability is to replicate data across the network while maintaining a sufficient level of their consistency. Besides boosting availability by increasing redundancy, another effect of replicating data on distributed sites is that "remote" accesses can be accomplished locally, thereby further improving accessibility and availability.

COPla ensures the consistency of transactions involving repicas. Solutions vary, depending on the kind of update

*This work has been partially supported by the EU grant IST-1999-20997 and the Spanish grant TIC99-0280-C02-01.

propagation used: eager or lazy. COPla supports different consistency protocols. Its API allows to deliberately switch from one consistency protocol to another, as needed. Moreover, each protocol supported by COPla may be run in any of three different consistency modes. Each session can change dynamically its consistency mode, thereby modifying the conflict handling rules for acceses to data, replicas of which are possibly accessed concurrently by other sesions, in the same or other database nodes. Thus, appropriate consistency guarantees can be chosen for satisfying the individual needs of each service.

Section 2 describes the structure and the functionality of COPla. Section 3 describes COPla's consistency management principles. Section 4 compares our approach, as related to eager replication protocols, with other systems. Section 5 summarizes the paper.

2. The COPla Architecture

The COPla architecture is structured by three layers which are interfaced via CORBA, so that each can be placed in a different node. Hence, a multitude of simultaneous services (running on different nodes) is enabled to access the same database replica. All updates applied to these replica are propagated to other database replicas using COPla's replication management components. The three layers, as depicted in figure 1, from bottom to top, are:

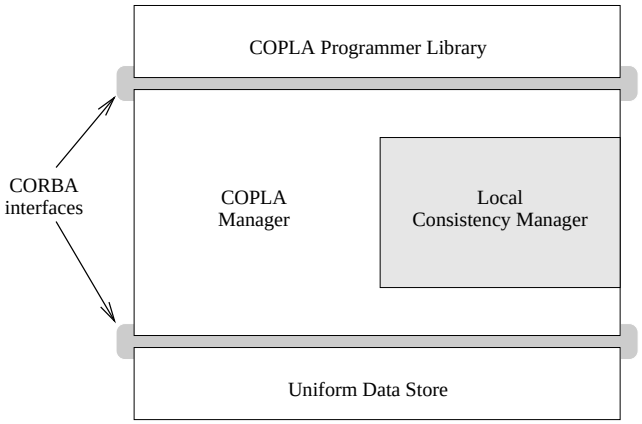

Figure 1. COPla architecture.

- *Uniform Data Store (UDS)*. This layer manages all persistent data. It interfaces with a relational DBMS, where the persistent objects of the given service and the metadata of the consistency protocol are stored. UDS isolates the upper layers from the actual storage system. Thus, the use of different brands of RDBMSs is supported. The schema definition of the databases uses GODL, a simplified version of the ODMG ODL language [3].

- *COPla Manager*. This layer is the core of COPla. It manages database *sessions* (which may include multiple sequential transactions, working in different consistency modes) and controls the set of database replicas comprised by the network. It also has some caches to improve the efficiency of database accesses. Moreover, it includes a *local consistency manager*, for handling several different consistency protocols. The supported protocols share some common characteristics. In general, all of the communication between the networked databases is controled by the local consistency managers.

- *COPla Programmer Library*. This is the layer used by applications to access system services. Applications, COPla manager and/or the UDS need not be installed on the same node; each only needs to have the library layer on their respective nodes. This layer also provides cache support and multithreading optimizations for improving the overall system performance.

3. Consistency Management

In 3.1 and 3.2, we describe general principles of COPla's consistency management. They are common to all consistency protocols which can be plugged into COPla. (Various eager and lazy protocols developed for our platform have been described in [11, 10, 7].)3.3 reports new performance measurement results for a replication protocol which attempts eagerly to restore consistency, details of which are described in [7].

3.1. Roles of Network Nodes

Given a session that tries to commit, the nodes involved in its execution may have two different roles:

- *Active node*. The node where the COPla Manager that has directly served the session's execution is placed.

- *Synchronous nodes*. All other nodes that have a COPla Manager. In these nodes, the session updates will be eventually received, if such updates exist. Note that read-only sessions do not generate any database updates. Hence, these sessions do not have any synchronous node.

Moreover, in a given session, multiple objects may have been accessed. Before committing a session, some checks have to be done to ensure that the accessed objects' states were up-to-date. One of the nodes receives a distinguished role in these checks, and the others will accept its decisions.

Consequently, for each object, there exists its *owner node*. That is the node where the object was created; it is the manager for the *access confirmation requests* sent by the active nodes at commit time. The management of these access confirmation requests is similar to lock management, but at commit time. To this end, the owner node compares two object versions, the one sent in the request (which is the object version accessed by the requesting session), and the latest object version that exists in the database. If they are not equal, the request is denied and the session will be aborted because it has accessed an outdated object version. On the other hand, if they are equal and there is no other granted request in a conflicting mode (a conflict exists if one of the requests comes from a session that has modified the object), a positive reply is sent to that active node. An active node can commit a session if all access confirmation requests that it has sent have been replied positively.

3.2. Consistency Modes

A session can be considered as a sequence of "transactions" done in the same database connection. Each of this "transactions" can be done in one of the following consistency modes:

- *Plain consistency*. This mode does not allow any write access on objects. It guarantees that all read accesses in this mode follow a causal order. On the other hand, this mode imposes no restriction on the currentness of the objects being read. Thus, they may be outdated.
- *Checkout consistency*. This mode is similar to the traditional sequential consistency, although it does not guarantee isolation. Thus, if several sessions have read a given object, one of these sessions is allowed to promote its access mode to "writing". However, if two of these sessions have promoted their access modes from reading to writing, one of them will be aborted.
- *Transaction consistency*. In this mode, the usual transaction guarantees are enforced: atomicity, sequential consistency, isolation and durability.

A session always starts in plain mode. If the guarantees provided in this mode are not sufficient for the given service, it can promote its consistency mode to checkout or transaction. In these two modes, all accesses are temporarily stored until an explicit call to the *commit()* or *rollback()* operations is done (with the usual meaning of such operations). Once one of these operations have been done, the session returns automatically to plain mode. Thus, the programmer is able to choose the consistency mode of each session of a given service, and this consistency mode can be changed as needed while a session is running.

3.3. Performance Measurements

The COPla architecture allows for an object-oriented view of a given database (i.e., collection of data objects), which is physically mapped to the replicated DBMS where the data are actually stored. Replication and consistency maintenance entail some additional burden on performance.

We have taken some measurements of these extra costs, for the eager FB consistency protocol. In particular, we have considered two types of transactions, with only two objects. The first one just reads the data objects, whilst the second one reads both objects and updates one of them (at random). The tests have used networks composed by one, two and four nodes. A singular node has been assigned to be the owner of both objects in all tests. In all test cases, only one session has been created in each node, and all transactions of that session have been executed sequentially. Complete sequentialization of transactions is the worst possible case, since the multi-threaded support and the caches are not used in that case.

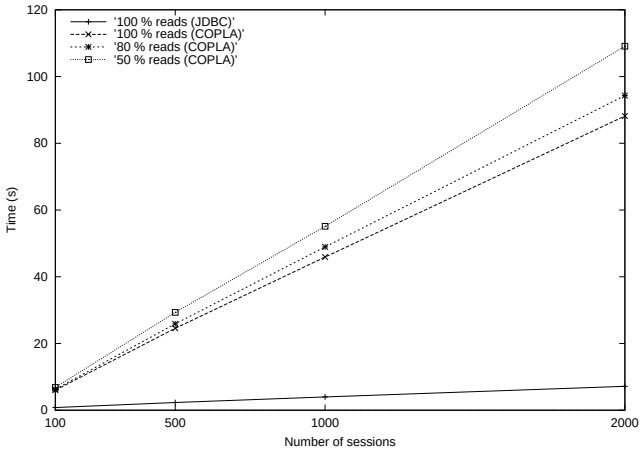

Figure 2. Elapsed times in a network with only 1 node.

Figure 2 shows the additional costs introduced by COPla. We compare the time required to execute a given number of transactions in a system consisting of a single node. So, the consistency protocol is not used here, but all COPla layers are actually used to map the object accesses in the service layer to the accesses needed in the relational database. Hence, this scenario is not contrived, but serves best to measure the overhead introduced by the COPla approach. As will become apparent, the overhead is proportionally reduced when the number of nodes increases.

Four lines are shown. The lowest one corresponds to the times needed when read-only transactions use directly the

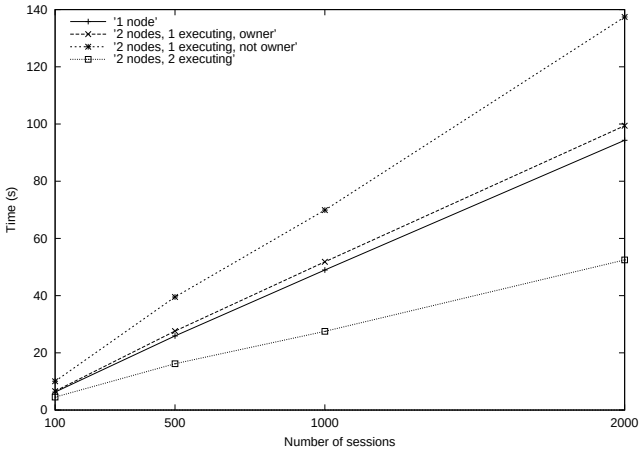

Figure 3. Elapsed times in different network configurations.

JDBC support; i.e., without COPla. The other three correspond to a load of read-only transactions, another with 80% of read-only transactions and the last one with 50% of read-only transactions, all of them using the COPla services. In this worst-case scenario, the read-only COPla transactions have a cost 10 times higher than the JDBC read-only transactions. However, we have to consider that currently, JDBC does not provide any replication support nor an interface easily usable in object-oriented programming (at least when it is compared to the interfaces provided by an ODMG-compliant platform such as COPla).

Figure 3 compares the results obtained from different system configurations with a mix of 80% read-only and 20% update transactions. We consider five different configurations. The first one uses only one node. All the others use a system with two nodes. In the second and third configurations, only one of the two nodes executes all the transactions, so the updates have to be transmitted to the passive node. The results vary, depending on the ownership of the objects. If all transactions have been executed by the owner node, the differences with the one-node configuration are minimal. However, if the transactions have been executed by the not-owner node, they require almost twice the time of the previous case, due to the access permission requesting and granting messages.

The last configuration correspond to a two-node system where both nodes execute transactions. In this case, the load is balanced between them and better performance can be obtained by COPla. Additionally, the overall time is lower than that of the one-node system.

It can be observed that the scalability of the FB protocol is well scalable (the system productivity is increased in 172% when two nodes are used in front to a single node system). Second and third lines also show that the overhead of ”non-ownership” with the FB protocol is below 30% in the evaluated environment.

We have taken other measurements with a 4-node network. When only one node executes the transactions and propagates the updates to the other ones, the measured times are equal to the 2-node network described above. This is the expected result, since the updates are broadcast, and its cost does not depend on the number of targets. On the other hand, when all the nodes directly execute transactions, the overall cost is reduced to approximately half of the time measured in the 2-node network, when both nodes have executed their sessions. This result is also reasonable, since the use of multiple nodes allows a better balancing of the system load.

4. Related Work

Current work in consistency protocols for replicated databases can be found using either eager [1, 9, 13] or lazy protocols [2, 4, 11]. Each has its pros and cons, as described in [5]. Eager protocols usually hamper the update performance and increase transaction response times but, on the positive side, they can yield serializable execution of multiple transactions without requiring too much effort. On the other hand, lazy protocols may answer read requests by stale data versions (or at least they require extra work to avoid that), but they improve transaction response times and allow disconnected operation.

COPla is a platform for both eager and lazy consistency protocols, but we focus on eager update propagation, in this section. In [13], a good classification of eager protocols is presented, according to three parameters: server architecture (primary copy vs. update everywhere), server interaction (constant vs. linear) and transaction termination (voting vs. non-voting). Only two of the eight alternatives resulting from combining the three parameters seem to lead to a good balance of scalability and efficiency: those based on “update everywhere” and “constant interaction”. This is mainly due to the load distribution achievable with the “update everywhere” approach, and the low communication costs result from a “constant interaction”.

The FB protocol complies with these two parameters. It uses “update everywhere” (instead of “primary copy”), because each transaction is done initially at the node where it was initiated, independent of the accessed objects. It also uses “constant interaction”, since the updates are only broadcast at transaction termination, once the object version checking has been done. Due to this version checking on the object owners’ nodes, the FB protocol must be classified as “voting termination”. Although “non-voting termination” approaches require less message rounds, they either need atomic reliable broadcasts, with total order delivery, if the

updates are done at commit time, or all nodes need to execute completely all transactions, even those that finally will be aborted (if the broadcasts are done when the transactions start). Thus, at first sight, a "voting termination" approach seems better.

However, our design differs a bit from the guidelines provided in [13] for the "voting termination" approach. Control of the transaction termination is based in our case on object versioning. Hence, the votes consist only in checking the accessed object versions, verifying that they have been the latest ones. We do not need a total order broadcast nor a 2PC to find out if a transaction is allowed to commit or not. Indeed, in the best case, we only need a single round of requests and answers to do the voting, and this round does not use a total order. Thus, our solution requires lower costs than those referenced as examples in [13].

5. Conclusions

Orthogonal to traditional approaches for enhancing database availability, the COPla architecture caters for the availability of data and services by supporting the consistency maintenance of replications over a multitude of network nodes. Within COPla, a range of different consistency protocols are provided. Depending on the needs of a given variety of application services, COPla users may choose from the set of available consistency protocols the one which fits best, in each particular case. Moreover, COPla supports three different consistency modes for each of its consistency protocols, which can be chosen at will for each transaction.

In this paper, we have described one of several consistency protocols that are available in the current version of our system. It is based on eager update propagation, but does not need a total order broadcast communication nor multiple update rounds. Hence, it minimizes the communication needs of such kind of protocols, thus reducing the usually long transaction completion time, which otherwise is one of the main drawbacks of eager protocols.

References

[1] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi. Exploiting atomic broadcast in replicated databases. *LNCS*, 1300:496–503, 1997.

[2] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, and A. Silberschatz. Update propagation protocols for replicated databases. *SIGMOD Record (ACM Special Interest on Data Management)*, 28(2):97–108, 1999.

[3] R.G.G. Cattell, D.K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez, editors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann Publishers, January 2000. 300 pgs., ISBN 1-55860647-5.

[4] F. Ferrandina, T. Meyer, and R. Zicari. Implementing Lazy Database Updates for an Object Database System. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 261–272, Santiago, Chile, 1994.

[5] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182, Montreal, Canada, 1996.

[6] Instituto Tecnol´ogico de Inform´atica. GlobData web site. Accessible in URL: *http://globdata.iti.es*, 2002.

[7] Luis Ir´un, Francesc Muñoz, Hendrik Decker, and Josep M. Bernab´eu-Aub´an. 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, April 2003.

[8] J.Esparza, A.Calero, J.Bataller, F.Muoz, H.Decker, and J.Bernab´eu. Copla - a middleware for distributed databases. In *3rd Asian Workshop on Programming Languages and Systems (APLAS '02)*, pages 102–113, 2002.

[9] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *International Conference on Distributed Computing Systems*, pages 156–163, 1998.

[10] Francesc Muñoz, Luis Ir´un, Pablo Gald´amez, Jos´e Bernab´eu, Jordi Bataller, and Mari-Carmen Bañuls. Flexible management of consistency and availability of networked data replications. *Flexible Query Answering Systems (FQAS '02)*, 2522:289–300, October 2002.

[11] F.D. Muñoz-Esco´ı, L. Ir´un-Briz, P. Gald´amez, J.M. Bernab´eu-Aub´an, J. Bataller, and M.C. Bañuls. GlobData: Consistency protocols for replicated databases. In *Proc. of the IEEE-YUFORIC'2001*, pages 97–104, Valencia, Spain, November 2001.

[12] P. Vicente and L. Rodrigues. An indulgent uniform total order algorithm with optimistic delivery. In *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, pages 92–101, oct 2002.

[13] M. Wiesmann, A. Schiper, F. Pedone, B. Kemme, and G. Alonso. Database replication techniques: A three parameter classification. In *Proc. of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, pages 206–217, October 2000.