COPLA: A Platform for Eager and Lazy Replication in Networked Databases.[*]

Francesc D. Muñoz-Escoí and Luis Irún-Briz and Hendrik Decker and Josep M. Bernabéu-Aubán
Institut Tecnològic d'Informàtica
Universitat Politècnica de València
Camí de Vera, s/n
46071 València, SPAIN
Email: {fmunyoz,lirun,hendrik,josep}@iti.upv.es

Key words: distributed databases, concurrency control, database replication

Abstract: COPLA is a software tool that provides an object-oriented view of a network of replicated relational databases. It supports a range of consistency protocols, each of which supports different consistency modes. The resulting scenario is a distributed environment where applications may start multiple database sessions, which may use different consistency modes, according to their needs. This paper describes the COPLA platform, its architecture, its support for database replication and one of the consistency algorithms that have been implemented on it. A system of this kind may be used in the development of applications for companies that have several branch offices, such as banks, hypermarkets, etc. In such settings, several applications typically use on-site generated data in local branches, while other applications also use information generated in other branches and offices. The services provided by COPLA enable an efficient catering for both local and non-local data querying and processing.

1 Introduction

Distributed applications may use replicated databases for taking advantage of a heightened availability enabled by a multitude of replicas of each data object. Some of such distributed applications are used by companies with multiple branches or offices that are distributed in a wide-spread area (national or international). Most of these companies distribute their information per branch, ensuring that the data are mainly updated locally, by the branch where they were created. Examples of such companies could be hypermarkets and banks (although very many of them still use a centralized solution).

Such applications mainly deal with data objects related to the local branch, but sometimes they require additional access to data objects created in other branches. If these accesses need to be done on a remote database, they could be inefficient. So, it seems advisable to replicate all data objects, independent of the company branch (and, in a sense, also of the local node embodied by the branch's database) where they have been created. Moreover, the majority of accesses made on non-local-branch data objects will be read-only accesses. If data are replicated, "remote" accesses can be accomplished locally, improving access times as well as availability, in case of node failures in the network). However, each time a data object is modified, updates have to be multicast to a given number of database replicas (not necessarily to all, depending on the replication technique being used), and this also needs some extra time. Hence, considering the system performance, database replication is only convenient if the larger share of accesses do not modify the data objects. But this limitation can be overcome if we also consider the availability benefits of having multiple replicas for each object. After all, in a WAN, node failures or network partitioning are not uncommon.

COPLA is a software platform in the GlobData project (Instituto Tecnol´ogico de Inform´atica, 2002) which supports database replication. It provides an object-oriented view of a network of relational DBMSs. The COPLA architecture is structured in three different layers, which interact using CORBA interfaces, enabling the placement of each layer in a different machine. Thus, multiple applications (running in different nodes) may access the same database replica using the COPLA support. All updates applied to these replica are propagated to other database

[*]This work has been partially supported by the EU grant IST-1999-20997.

replicas using the replication management components of COPLA.

One of the problems solved by COPLA is to ensure the consistency among transactions being executed in different database replicas. The solution to this problem depends mainly on the update propagation model: eager or lazy. COPLA is flexible enough to allow multiple update approaches, each one being managed by a dedicated consistency protocol. A particular COPLA component implements the consistency protocol. Multiple implementations of its interfaces are possible, thus allowing that a GlobData system could change its consistency protocol at will.

Moreover, all consistency protocols support three different consistency modes that can be used by any database session. Each session can change dynamically its consistency mode, thereby modifying the conflict handling rules for its accesses to data objects concurrently being accessed by other sessions in the same or other database replicas. So, a programmer who uses the COPLA support disposes of a rich toolset from which appropriate guarantees can be chosen for satisfying the needs of the application to access its data objects.

This paper provides an outline of COPLA and a detailed description of one of its "*eager update*" GlobData consistency protocols (lazy and other GlobData protocols have been described elsewhere (Muñoz-Escoí et al., 2001)). To this end, the following section describes the structure and the functionality of COPLA. Section 3 explains the GlobData-FB consistency protocol. Section 4 compares this protocol with other systems. Finally, section 5 gives some conclusions.

2 The COPLA Architecture

The COPLA architecture consists of the three layers as depicted in figure 1. From bottom to top, they are the following:

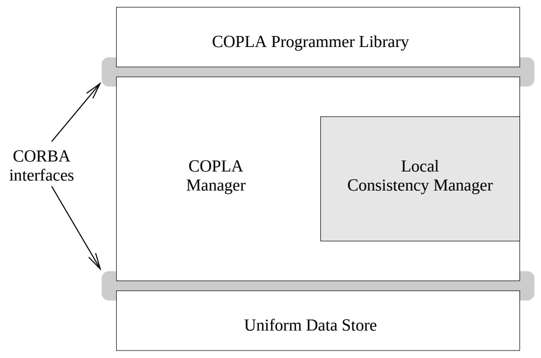

Figure 1: COPLA architecture.

- *Uniform Data Store (UDS)*. This component manages the persistent data of a GlobData system. It interacts directly with a relational DBMS, storing there the persistent objects of the given application and the metadata of the consistency protocol. It isolates the upper layers from the actual storage system used. In practice, support for different RDBMSs will be provided in the final release of the UDS (currently, it only manages PostgreSQL repositories).

  The definition of the application databases is made using GODL, a simplified version of the ODMG ODL language (Cattell et al., 2000).

- *COPLA Manager*. The COPLA manager is the core component of the COPLA architecture. It manages database *sessions* (which may include multiple sequential transactions, working in different consistency modes) and controls the set of database replicas that compose the GlobData system. This manager also provides some caches to improve the efficiency of the database accesses.

  A *local consistency manager* is included in this layer. Multiple consistency protocol objects may be used in this component, but only one is allowed at a time. All consistency protocols share some characteristics. For instance, all of them are rather optimistic, since no locks are used "a priori" to request the objects being accessed. Hence, the consistency checks must be done at commit time. If a session is allowed to commit, its updates are multicast by its local consistency component to all consistency components placed in other GlobData nodes. The way this is done depends on the consistency protocol being used. All of the communication among GlobData databases is managed by this component.

- *COPLA Programmer Library*. This library is the layer used in GlobData applications to access system services. It also provides some cache support and multithreading optimizations that improve the overall system performance.

  The applications need not be installed in the same node where the COPLA manager or the UDS are placed. They only require this library layer on their nodes.

Between each pair of consecutive layers, there is a CORBA interface. So, each layer could be placed in a different node. To enable communication across layers, an object request broker (ORB) is needed. The current system release is implemented in Java, and the Java ORB of the Sun J2SDK is used.

3 GlobData-FB Consistency Protocol

The GlobData-FB consistency protocol uses "*Full Broadcast*" of the session updates, once the session is allowed to commit. It is one of the protocols available within COPLA for consistency management (others are described in (Muñoz-Escoí et al., 2001)). It is described in 3.3; further clarifications are presented in the subsections thereafter.

3.1 Node Roles

Considering a given 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 made 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" made in the same database connection. Each of this "transactions" can be made 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 made 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: atomicity, sequential consistency, isolation and durability, are enforced.

A session always starts in plain mode. If the guarantees provided in this mode are not sufficient for the application, 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 made (with the usual meaning of such operations). Once one of these operations have been made, the session returns automatically to plain mode.

Thus, the programmer is able to choose the consistency mode of each session that composes her or his application, and this consistency mode can be variegated as needed while a session is running.

3.3 Protocol

As described above, the GlobData-FB consistency protocol broadcasts object updates to all synchronous nodes when a session is allowed to commit, thereafter. Consistency conflicts among sessions are resolved using object versioning. To this end, the protocol uses some metadata tables in the database where the current object versions can be stored.

The protocol processes the following steps:

1. In active nodes, sessions are created and executed without any additional check. They are allowed to proceed until they request their commit operation.
2. When an application tries to commit one of its sessions, such operation arrives at its COPLA manager, and before applying the commit to the associated UDS, the local consistency manager is informed thereof. To this end, the COPLA manager builds two sets containing the identifiers of all objects read and written in such a session. These are the *session read-set* and *session write-set*.

3. Once these sets have been received by the local consistency manager, the latter sends an *access confirmation request* to the owner of each of the objects in the sets. Such messages include the object identifiers, their accessed versions, the access modes (read or write) and the consistency mode used by the session (checkout or transaction, since plain mode does not need a commit operation).

4. The owner node of each object checks if this access confirmation request would conflict in any way with previous access confirmation requests granted to other sessions but not yet released. A conflict arises if the requesting session has written the object and there is another session that has previously obtained a write grant on the same object version.

   Additional conflicts depend on the consistency mode of the sessions involved in the check. If all sessions have used checkout mode, then a conflict only arises if the requesting session has modified the object and other read-access grants have been obtained previously by other sessions. But in checkout mode, a session does not run into conflicts by having read outdated object versions.

   On the other hand, if at least one of the currently committing sessions has used transaction mode, then conflicts arise when either the requesting session has used an outdated object version, or when there are multiple sessions accessing the object and at least one of the sessions has written it.

   If the owner finds that a conflict arises, then it answers the access confirmation request with a *deny* reply. Otherwise, it sends a *grant* reply and the session identifier is recorded as "granted" until it explicitly releases this grant in step 5 or 8.

5. When the active local consistency manager receives the replies, and if at least one reply denies the access confirmation requests, then the session is aborted. However, if all of them grant the request, then the session will commit.

   If the session has been aborted, then a release message is sent to the object owners that had replied using a "grant" message.

6. If the session has been allowed to commit in the previous step, then the consistency manager of the active node broadcasts the session updates to all GlobData system nodes; i.e., to all nodes that have a consistency manager. This is an atomic broadcast.

7. Once the update message is received, the active node for that session commits it. The synchronous nodes will also commit the session updates. But before doing so, they have to check that no local session has accessed any of the objects received in that update message. If such local sessions exist, they are aborted.

8. Once the update has been completed, the consistency managers placed in the synchronous nodes check if they are the owners of some of the objects updated. If that is the case, the grants set in step 4 are released immediately. Since no explicit message is needed to do so, this accomplishes the protocol.

3.4 Fault Tolerance

Since data objects are replicated in several RDBMSs, COPLA is able to tolerate failures of part of the system nodes. To this end, the following protocol details must be considered.

- *Session completion*. If the active node of a session fails before it completes the atomic broadcast of the session updates, the occurrence of such a session is unknown on the rest of nodes. Hence, the session has to be aborted when the active node recovers.

  However, if the update atomic broadcast has been completed, the session has been committed on all system nodes. In this case, the only node that probably has not committed that session is the active one. But this does not matter, since the session updates will be transferred to that node when it recovers, if needed.

  Another undesirable effect of this kind of failure is related to the access grants that the session may have obtained. Since the session has not been committed, these grants would remain assigned to the faulty node, preventing other sessions from obtaining access to such objects. The solution to this problem is easy. When the access confirmation requests have been received by an owner node, the identifier of the node that has made such requests is memo'ed and associated to the requests in some data structure of the owner. When the membership service notifies that a node has failed, this data structure is scanned and all grants assigned to it are automatically released.

- *Ownership role migration*. When a node fails, all objects that were created in there have lost their owner. To replace it, the node with the next identifier in increasing order is chosen as a *temporary owner* for such objects. This temporary owner retains its role until it also fails and is replaced by another one, or until the original owner recovers.

  Moreover, some steps are needed to obtain the lists of access grants that the faulty owner had when it failed. COPLA uses a membership service that notifies all live nodes about any membership change (either join or failure). When such a notification is received, each consistency manager scans its list of received access grants and builds a message containing all the information of all grants given by the faulty owner. This message is then sent to the temporary owner that will replace the faulty one. If a node does not hold any grant of this kind, it must send an empty message.

The temporary owner collects all such messages and builds a list of its granted confirmation requests. New access confirmation requests are not replied until such a list has been rebuilt.

- *Node recovery*. The recovery steps needed when a node rejoins the GlobData system are thoroughly described in (ITI and FCUL, 2002). An outline of these steps is provided subsequently.

1. Once a given node has failed, all remaining live nodes memorize the OIDs of all objects updated in all sessions committed since then, and associate these notes to the faulty node identifier, i.e., they add the OIDs to a hash table or some data structure which is indexed by node identifiers. Hence, all live nodes have registered the same state.
2. Once a faulty node recovers and tries to join again the GlobData system, all live nodes freeze their respective databases. To this end, if some session has started the commit protocol or has modified at least one of the objects owned by its local node, it is allowed to terminate. Other sessions are blocked until the joining node is integrated in the system.
3. Once the allowed sessions have terminated, the live nodes send a message to the joining one, communicating that their local databases are prepared for the joining procedure. The joining node waits for all such messages. When it has received all of them, it broadcasts a message to these nodes, indicating that it expects the database updates.
4. The aforementioned nodes of the GlobData system reply to this message by another one, including the contents of all objects whose OIDs can be found in the hash table described in the first step and that are owned by the replying node. Hence, the database updates are collected from different nodes.

   Note that this recovery protocol is used in all consistency protocols outlined in (Muñoz-Escoí et al., 2001). Although the distributed collection of database updates as described in the previous paragraph does not provide any advantage or inconvenience for the GlobData-FB consistency protocol, it permits a fast collection when the updates are propagated lazily. Instead of using another approach for the GlobData-FB case, we prefer to use the same recovery steps as in the lazy consistency protocols.
5. Later, the joining node applies all such updates using a newly created transaction, which will be committed when the whole set of replies has been received.
6. Finally, the joining node broadcasts another message, indicating that the joining process has concluded and allowing the blocked sessions to go on.

3.5 Performance Measurements

The COPLA architecture provides a mapping from an object-oriented view of a given database to the replicated DBMS where the data is physically placed. Moreover, it provides support for data replication. As a result, this support implies some performance costs in the system services.

We have taken some measurements of these extra costs. To this end, we have considered two types of transactions used on a database where only two objects exist. The first one reads all the database objects, whilst the second one reads both objects and updates one of them (at random). The tests have used a GlobData system composed by one, two or four nodes. One node is the owner of both objects in all tests. In all cases, only a session has been created in all nodes, and all the transactions have been executed sequentially in that session. This is the worst case in our system, since the multithreaded support and the caches are not be used in this system arrangement.

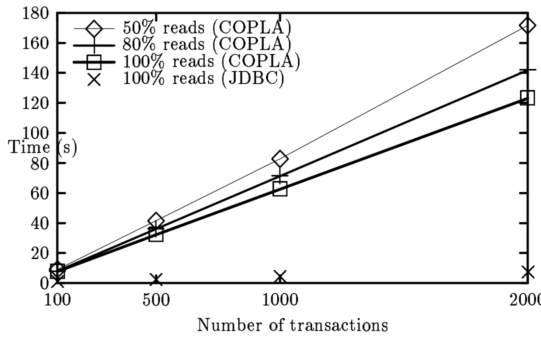

Figure 2: Elapsed times in a GlobData system with only 1 node .

Figure 2 shows the additional costs introduced by our COPLA support. We compare the time required to execute a given number of transactions in a system composed by only one node. So, in this environment the consistency protocol is not used, but all COPLA layers must be used to map the object accesses in the application layer to the accesses needed in the relational database.

Four lines are shown. The lowest one corresponds to the times needed when read-only transactions use directly the JDBC support; i.e., without COPLA. The

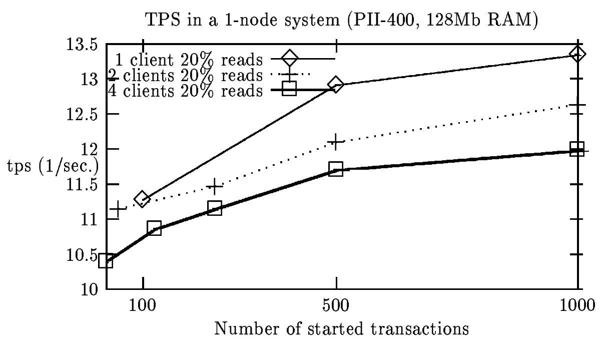

Figure 3: Load delivery in a minimal GlobData system configuration.

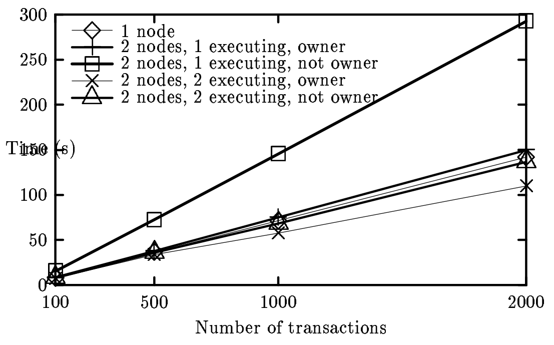

Figure 4: Elapsed times in different GlobData system configurations.

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. The read-only COPLA transactions have a cost 9 times greater 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 system like COPLA).

Another interesting measurement performed can be observed in figure 3. There is shown the evolution of the transactions commited per second in a minimal GlobData system. This scenario is composed by a single node, where multiple clients access to a common repository. The set of sessions initiated by these clients has been modelated as mix of 80% read-only and 20% update transactions. The database used to this experiment contains only four elements, in order to increase the effects of the concurrency of the clients.

The experiments shows how the number of transactions commited in the system decrease when the number of clients grows up. This is caused by two reasons: overhead introduced by the concurrency control, and the cost of the aborted sessions.

The figure shows the system trend to reach an stable situation, in which the aborted sessions balance the load of the system.

Figure 4 compares the results gotten from different system configurations when a mix of 80% read-only and 20% update transactions are used. 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 two configurations correspond to a two-node system where both nodes execute transactions. In this case, the load is balanced between them and there is not an appreciable difference between the owner and not-owner nodes. Additionally, the overall time is lower than that of the one-node system.

We have taken other measurments with a 4-node system. When only one node executes the transactions and propagates the updates to the other ones, the measured times are equal than in the 2-node system described above. This is the expected result, since the updates are broadcast and such a broadcast 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 a half of the time measured in the 2-node system when both nodes executed the sessions. This result is also reasonable, since the use of multiple nodes allows a better balancing of the system load. [1]

4 Related Work

Current work in consistency protocols for replicated databases can be found using either eager (Agrawal et al., 1997; Kemme and Alonso, 1998; Wiesmann et al., 2000) or lazy protocols (Breitbart et al., 1999; Ferrandina et al., 1994; Muñoz-Escoí et al., 2001). Each has its pros and cons, as described in (Gray et al., 1996). Eager protocols usually hamper

[1]We are planning to include the abort rates in the final version of the paper, but currently we are unable to provide them due to a bug in the protocol implementation.

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.

Although COPLA provides a framework able to accept different consistency protocols (indeed, lazy protocols have also been designed for this environment (Muñoz-Escoí et al., 2001)), the presented approach uses an eager replication alternative. A good classification of eager protocols is presented in (Wiesmann et al., 2000), according to three parameters: server architecture (primary copy vs. update everywhere), server interaction (constant vs. linear) and transaction termination (voting vs. non-voting). Among the eight alternatives resulting from combining these three parameters, only two of them 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, i.e., a delegate server executes the transaction and broadcasts the changes everywhere. The election of such a delegate server is dynamic. Each transaction can choose a different delegate. Moreover, low communication costs result from a "constant interaction", where the update broadcast is done just once, either at the beginning or end of the transaction, rather than in each transactional operation, as is the case in the "linear interaction" approach.

The GlobData-FB protocol shares these two characteristics. It uses "update everywhere" (instead of "primary copy"), because each transaction is done initially at the node where it was initiated, independently of the objects being accessed. It also uses "constant interaction", since the updates are only broadcast at transaction termination, once the object version checking has been made. Due to this version checking on the object owners' nodes, this 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 made at commit time, or all nodes need to execute completely all transactions, even those that finally will be aborted (if the broadcasts are made when the transactions start). So, at first sight, a "voting termination" approach seems better.

However, our design differs a bit from the guidelines provided in (Wiesmann et al., 2000) 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. So, our solution requires lower costs than those referenced as examples in (Wiesmann et al., 2000).

Finally, although our technique provides good results in terms of communication needs (i.e., delivery ordering and number of messages), the use of a communication tool that provides total order guarantees may simplify the recovery protocols when failures occur. Indeed, the recovery protocols described in (Kemme et al., 2001) allow that the previously running nodes do not block when a faulty node recovers. This advantage currently is not available in our system.

5 Conclusions

The COPLA architecture provides an object-oriented view of data items in a network of distributed and replicated relational databases. This architecture establishes a framework where it is easy to install different consistency protocols. Depending on the needs of a given volume of applications, the COPLA users may choose among the set of available consistency protocols the one which fits best for these applications.

Moreover, COPLA provides support for three consistency modes in each of its consistency protocols. So, the application programmer may also choose which consistency mode may fit best for each sequence of accesses, for a given application.

In this paper, we have sketched the COPLA architecture and described one of several available consistency protocols in the GlobData system. It is based on eager update propagation, but it does not need a total order broadcast communication nor multiple update rounds. Hence, it minimizes the communication needs of such a kind of protocols, reducing in this way the usually long transaction completion time, which is one of the main drawbacks of eager protocols.

REFERENCES

Agrawal, D., Alonso, G., El Abbadi, A., and Stanoi, I. (1997). Exploiting atomic broadcast in replicated databases. *Lecture Notes in Computer Science*, 1300:496–503.

Breitbart, Y., Komondoor, R., Rastogi, R., Seshadri, S., and Silberschatz, A. (1999). Update propagation protocols for replicated databases. *SIGMOD Record (ACM*

Special Interest Group on Management of Data), 28(2):97–108.

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

Ferrandina, F., Meyer, T., and Zicari, R. (1994). 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.

Gray, J., Helland, P., O'Neil, P., and Shasha, D. (1996). 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.

Instituto Tecnol´ogico de Inform´atica (2002). GlobData web site. Accessible in URL: *http://globdata.iti.es*.

ITI and FCUL (2002). Implementation of the scattered data manager. Technical report, D07, GlobData Working Group, IST Programme, project number: IST-1999-20997.

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

Kemme, B., Bartoli, A., and Babaoğlu, Ö. (2001). Online reconfiguration in replicated databases based on group communication. In *Proceedings of the Internationnal Conference on Dependable Systems and Networks (DSN2001)*, Göteborg, Sweden.

Mu˜noz-Esco´ı, F., Ir´un-Briz, L., Gald´amez, P., Bernab´eu-Aub´an, J., Bataller, J., and Ba˜nuls, M. (2001). GlobData: Consistency protocols for replicated databases. In *Proc. of the IEEE-YUFORIC'2001*, pages 97–104, Valencia, Spain.

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