# Recovery Protocols for an O2PL Consistency Protocol*

J.E. Armendáriz, J.R. González de Mendívil
Dpto. de Matemática e Informática
Universidad Pública de Navarra
Campus Arrosadía s/n, 31006 Pamplona, Spain
{enrique.armendariz, mendivil}@unavarra.es

F.D. Muñoz-Escoí
Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
Camino de Vera s/n, 46071 Valencia, Spain
fmunyoz@iti.upv.es

## 1. Introduction

The increasing business globalization, and the growing diffusion of Internet and of Wide Area Networks (WANs) in general, creates for corporations and institutions the need to store very large amounts of data in distributed databases that span buildings, cities, countries and even continents. These data must be accessed by users, also geographically scattered, that must consult or work with them, very often in a concurrent way. This sharing and interaction is not well handled by traditional data base methods, and requires new software engineering processes.

We will develop our Optimistic Two-Phase Locking (O2PL) consistency protocol [2], along with its corresponding recovery protocol –that has never been described by its original authors–, in a replicated architecture called GlobData [4]. The overall objective of GlobData is to design and produce an efficient software development tool and support system called COPLA, to provide application developers with a global view of an object database repository. It provides a transactional access to geographically distributed persistent object repositories, no matter where they are physically located. Therefore, it reduces the traditional bottleneck of remote access, allowing application developers to efficiently work against a single logical object environment, although the actual objects are geographically distributed.

We consider a *fail-stop* and *primary partition* system model. In this context we propose two recovery protocols. The first one mixes the advantages of logs and object states transfer. It stores both, logs of transactions performed while a node was down and object identifiers (OIDs) modified by these transactions. A threshold is set up so as to neglect the logs due to its size and to start object state transmission for those objects modified while a node was down. The second protocol serves local transactions in the recovering replica even in the recovery process, but it cannot use log transfers. Additionally, in both cases the extra recovery workload can be distributed among all the alive nodes so all nodes are equally penalized, and none of our algorithms needs to block any of the non-recovering replicas, following an approach similar to that described in [3]. A multi-version concurrency control solution is assumed in the underlying DBMS, and this guarantees that the OID-based algorithm does not block the non-recovering replicas.

In the following, we will depict the consistency protocol and the recovery protocols used in our system.

*This project has been supported by the CICYT under research grant TIC2003-09420-CO2.

## 2. Optimistic Two-Phase Locking in COPLA

We have developed an implementation of the O2PL originally proposed by Carey in [2]. In fact, it is an optimistic version of the Two Phase-Locking (2PL) described by Bernstein et al. in [1]. It handles read requests and deadlock detection exactly the same way 2PL does. However, they differ in how replicated data items are treated. When a cohort updates a data item, its write request is managed locally, but it defers requesting locks on remote copies until the commit phase is reached. Since locks on copies are delayed until end of transaction, blocking and deadlocks may become rather late in the execution of a transaction. It defines a new kind of locks, called *copy-lock*s, which behaves the same way as an ordinary write-lock and speeds up the detection of deadlocks.

## 3. Recovery Protocols for O2PL in COPLA

Here is sketched a first draft of our recovery algorithms proposal. We propose two different alternatives, which provide different parameters settings, such as the transmission of logs missed by the faulty node or, otherwise, the transmission of the state of objects modified while the failed node was out. This first alternative will block the recovering node; i.e., it will not accept local transactions processing until the end of the recovery process.

The second approach will only store OIDs that have been modified while the faulty node crashed. Once the node is raised again, it will receive all the updates from all the alive nodes (this workload will be uniformly distributed). At the same time, it will allow to create new local transactions, that will be aborted either when an update request comes or at commit time if the local transaction accesses or updates an outdated object.

### 3.1. Switching Between Log and Object Transfer

An outline of this algorithm can be described in the following steps:

1. Alive nodes will not request copy-locks on the recovering node until the recovery process is done.
2. Thus, the transaction updates and the OIDs are stored by all alive nodes, which will be transmitted once the node rejoins the system.
3. Once a node is raised, all the transaction updates will be sent to the faulty node[1]. Meanwhile, all transaction updates generated by active nodes are stored too, as long as the recovery process takes place. This will continue indefinitely until the log size is smaller than a given value.
4. In the last transfer, a flag is set notifying the recovering node about the end of data transfers.
5. Then, the recovering node is considered to be alive (it broadcasts an "I am alive" message) and locks can be requested following the ordinary procedure. It will not answer to this lock requests until all pending transaction updates have been applied.
6. A node being recovered will not accept local transactions until the previous step has finished.

### 3.2. Non-Blocking on Recovering Node

As it has been previously depicted, this recovery protocol stores OIDs updated by transactions while a node has crashed. This approach avoids multiple update transfers on the same object, reducing them to the latest one. The algorithm is described hereafter:

1. Whenever a node fails the rest of alive nodes store the OIDs subsequently modified.
2. Once the faulty node is up, the rest of nodes deliver the state of objects modified. At the same time, it can immediately start local transactions following a special policy:
   (a) If a local transaction is accessing an object that is going to be updated, it will be aborted.
   (b) Otherwise, it reaches the commit phase. The read and write locks acquired are sent to an alive node, it checks whether these objects are up to date (searching them in the set of OIDs to be transferred to the recovering node), otherwise these lock requests are rejected and the transaction is aborted. If it is not aborted it can request copy-locks on the rest of alive nodes.
3. The end of this algorithm uses steps 4 to 6 of the previous protocol.

[1] We will also consider object state transfer instead of transaction updates whenever the amount of data is considerably large.

## 4. Conclusions

This extended abstract deals with the definition of recovery algorithms to be implemented in a replicated middleware architecture. This architecture utilizes several consistency protocols, one of them is the O2PL proposed by Carey et al. in [2]. We have outlined two recovery algorithms for this consistency protocol. The first one may use either a log- or OID-based update transfer to the recovering node. This may speed up the recovery process, since logs are a better solution in case of short-term crashes, but objects transfers are more adequate in the long-term case.

The second recovery algorithm allows local transactions service in the recovering node. Usually this may lead to a high abortion rate, but this solution can be optimal for applications with a high degree of locality, at least for update accesses.

## References


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

[2] M.J. Carey and M. Livny. Conflict detection tradeoffs for replicated data. *ACM Trans. on Database Sys.*, 16(4):703–746, December 1991.

[3] R. Jiménez-Peris, M. Patiño-Martínez, and G. Alonso. Non-intrusive, parallel recovery of replicated data. In *Proc. of 21st Symposium on Reliable Distributed Systems*, pages 150–159, Osaka Univ., Suita, Japan, October 2002. IEEE-CS Press.

[4] F.D. Muñoz Escoí, L. Irún-Briz, P. Galdámez, J.M. Bernabéu-Aubán, J. Bataller, and M.C. Bañuls. Globdata: Consistency protocols for replicated databases. In *Proc. of the IEEE-YUFORIC'2001, Valencia, Spain*, pages 97–104, November 2001.