Design of the PerDiS Transactional File System

João Garcia, Paulo Ferreira, Paulo Guedes

{joao.c.garcia, paulo.ferreira, paulo.guedes}@inesc.pt

INESC / IST[1], Lisboa, Portugal

September 1997

Abstract

We present the design for a fault-tolerant persistent data storage in a wide-area distributed cooperative engineering environment. We discuss some of the current options in file system design and, in particular, the relation between file mobility and transaction protocols. We believe that file mobility, and consequently coherence protocols, global file naming and versioning, are issues whose exploration and use will lead to improved transaction protocols and to an increased applicability of transactional file systems.

1.Introduction

Our aim is to implement a wide area distributed and fault tolerant file system in the context of the current PerDiS (Persistent Distributed Store) ESPRIT project[SHA97]. The emphasis of our design concerns is placed on improving the transactional and fault tolerant behavior of distributed file systems.

Several issues have been investigated in relation with distributed fault tolerance such as file naming, transactions, checkpointing, recovery, and reduction of dependency on remote machines. A distributed file system that is to provide persistent, fault-tolerant storage will have to consider these aspects. In the following sections we discuss how these techniques can be combined in order to create a file system, the PerDiS FS, with improved transactional performance as support for a distributed cooperative engineering environment.

2.Basic PerDiS FS Abstractions

The data in our persistent distributed store consists of object graphs which are made persistent according to the model of persistence by reachability[ATK83]. These object graphs are grouped in clusters andstored in files. The PerDiS FS will see these files as byte arrays. Files may include references, which are not language level references but the result of a swizzling mechanism. These stored references refer to other objects in the same file (file offsets) or in other files (filenames).

The PerDiS FS will use a name interface. The definition of the name space is always an important part of a file system's design. Two major options are choosing between location dependent and independent names and choosing between efficient but unintelligible names (e.g. file identifying numbers) and cumbersome but suggestive names (e.g. URL).

Location dependent file names are useful in systems where files are always located in the same machine. When one allows files to be moved, these names can still be used but no longer contribute to locate files directly. Therefore, movable files require inevitably a name/location translation mechanism.

In object based storage the majority of file use is system initiated. In a system where there are inter-file references, it is expectable that the majority of reference resolutions will result from graph traversal and not from user requests. For example, when browsing Web documents, it is more usual for users to follow links interactively than to type URLs.

---

[1] Instituto de Engenharia de Sistemas e Computadores / Instituto Superior Técnico

URLs are the current file identifiers in PerDiS. The arguments above and the use of file mobility (discussed below) suggest that speed in name/location resolution is a top priority in designing the name space for the PerDiS FS. It is natural that users wish to use more intelligible names, e.g. URLs, but this should be placed in a level above the name/location resolution mechanism. As we will see in section 5, versioning also poses serious challenges to the choice of a file name space.

Inter-file references are becoming more and more common, are actively used in Web documents (URLs) and are crucial in both PerDiS and other persistent systems, e.g. Pjava[ATK96][JOR]. An object that maintains a link to another object in another file must store that filename and possibly access the other file. Nevertheless, there are no techniques to provide file system support to these references.

Open for research is the question of whether the support for following inter-file references (filenames inside files) and maintaining them in a coherent state should be provided by the FS or by another level above it. Including this kind of support inside the FS is likely to lead to a modification of the usual structure of files by adding metadata that enables the location of references within files. The result is likely to divert too much from the concept of a file system.

One of the problems in using persistence by reachability is the size of the object graph accessible from one persistence root. Typically an object refers to other objects and therefore when one wants to persistently store an object one should also store the objects referred by it. The issue here is how far this recursive reasoning should be extended. The only existing solution to restrict a reference following explosion is to allow the user to indicate which references should or should not be followed when storing a graph of objects. This user-dependent method destroys the system's transparency but can be efficient if its use results in a definition of the graph's limits thereby defining a persistence space.

The path and depth with which an object graph is traversed is relevant also for data locking. What is a good conservative policy for pessimistic locking of object graphs? On the one hand, locking all reachable objects is certainly excessive. On the other hand, conflicting locking attempts by various transactions lead to more complex problems such as the need for versioning (see 5) and reconciliation of concurrent non-pessimistic transactions.

Until more satisfactory solutions to the problem of defining the limits of reachability spaces aren't found, the PerDiS FS design will only provide operations on files (or parts thereof).

3.File Mobility

One of the major sources of failure in a wide-area distributed application is the unavailability of remote data. This is generally caused by communication or node failure. Therefore, one of the contributions to reducing the dependency on remote machines is copying or moving the data kept therein closer to a local machine.

Transactions and increasing the locality of data are two issues that are closely related. Being able to access data locally increases not only its availability but also decreases the latency in accessing it. Reduced latency is crucial for the execution of distributed transaction termination protocols which is another essential point in distributed systems fault-tolerance. One of the main criticisms of this type of protocols is that the large number of communication steps they involve considerably slow down data access. These communication delays disappear when the transaction is performed on data stored locally.

4.Transaction Models and Coherence Protocols

Transactional access to data requires the definition of a transaction model. When we access a set of data, different transaction models require the locking of different data subsets in various lock/unlock sequences.

One can envisage that different applications will use objects requiring varied coherence guarantees and will therefore access them with different types of transactions [GRA93](e.g.: optimistic, pessimistic, etc...).

When one wishes to keep files in (or as close as possible to) the machines using them, this will generally result in the need for a coherence protocol. Coherence protocols have been widely studied in distributed shared memory and shared memory multiprocessors, but are, only now, beginning to be used in distributed file systems.

The choices of a transaction model and a coherence protocol are independent but influence each other's performance since coherence protocols change the location of current valid file versions and these are needed to perform commit protocols.

The most accepted and understood transaction type are ACID two-phase-locking (2PL) pessimistic transactions. At the moment the transactional interface provided by PerDiS has a standard format, essentially: begin_transaction, commit, abort and renew_transaction where all open files are included in transactions [FER97].

We plan to choose a coherence protocol that optimizes ACID 2PL transactions by means of bringing the data closest to the node coordinating the transaction. From there on, we will explore the possibilities of including other, more relaxed transaction models which will likely result in less demanding coherence protocols. We intend to incorporate PerDiS transaction programming interface in the PerDiS FS and the inclusion of new transaction models will certainly cause the extension of that interface.

5.Checkpoints and Snapshots

A transactional file system provides a model of persistence. Traditional file systems provide a "last writer wins" semantic to write accesses. A transactional file system has more semantic possibilities and there are several models that explore them, e.g. pessimistic transactions, optimistic transactions, etc…

The PerDiS FS should support different storing methods. Usually, a file system provides functionality for updating existing files, i.e. a checkpoint of working data, but the creation of new versions of existing files, snapshots, must be performed explicitly by applications.

The main option presented in this respect is whether to support versioning or not: Should the PerDiS FS be able to, explicitly (by user request) or implicitly (due to synchronization conflicts), create new versions of a file? This would provide significant assistance to application programmers but poses serious problems.

Versioning adds another dimension to synchronization, i.e. avoiding conflict by creating multiple copies of data. Versioning is very adequate for cooperating environments where different tasks or even different executions of the same task result in conflicting versions of the data. File system support for versioning would be useful for both applications and conflict resolution protocols.

Both checkpoints and snapshots are performed in the context of a transaction and might involve updating more than one file and here the problem of inter-file references reappears. Writing checkpoints ("saving" a set of files) poses no problem but snapshooting the state of a transaction ("save as" of a set of files) involves creating new versions of all the files with new filenames and consequently changing the filenames that exist as inter-file references within those files. This is a significant hurdle in providing this type of support.

Other distributed file systems have tackled the problem of checking for dependencies between concurrent accesses (generally, write-write conflicts) and using some criteria to resolve them [TER95]. Nevertheless, most of them block in the face of unsolvable conflicts and notify the programming level above. Versioning criteria are a good path to follow in order to extend conflict resolution algorithms.

6.Recovery

The main difficulty posed by wide-area data management is the scalability of the file system's metadata. This is the rationale for some of the latest distributed file systems such as xFS [AND95][WAN93] that takes the approach of organizing, the system's nodes in hierarchical

structure so as to reduce to size of the metadata. File systems nodes maintain a combination of global metadata, which ends up being replicated, and some local metadata. Therefore, the base for node failure recovery is a combination of checkpointing local metadata and cooperatively caching replicated metadata. A particular recovery algorithm depends on the metadata existing in a design of the system and can only be proposed after the design is concluded and the system's metadata are clearly defined.

7.Security

Typically, wide area file system may include machines in separate security domains using unsafe network connections. Guaranteeing that circulating data is ciphered is important for users with and, from a fault tolerance perspective, authentication is essential to ensure that the file system's coherence isn´t corrupted by rogue control messages. Therefore, if there is to be cooperation between unrelated nodes, security mechanisms must be included in the file system to authenticate the intervening nodes and cipher the file systems messages. Another relevant matter Finaly, in cooperative environments, file mobility and security are not independent matters. Moving and copying files among the system's machines depends on authorization from each file's home site. An entity providing a file for some remote manipulation will certainly require some guarantee that it will eventually recover the updated file and its ownership.

8.Conclusion

In this paper, we discussed the issues involved in designing the fault-tolerant transactional distributed PerDiS FS and described some of the options available.

In summary, the essential features to provide fault-tolerance and good performance for the PerDiS FS are:

- Hierarchical organization to ensure metadata scalability
- Efficient name/location resolution for fast file access.
- Movable files for greater independence from remote machines and for improvement of commit protocols performance.
- Versioning as a tool for applications and for transaction conflict resolution.
- Flexible transaction protocols with support for conflict resolution.

In particular, the performance dependency between file mobility and transaction protocols is a tradeoff which we believe can be a source of significant performance improvements and will be the subject of future research.

9.References

[AND95] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, Randolph Y. Wang. Serverless Network File System. *In* SIGOPS' 95. Colorado, USA. December 1995.

[ATK83] M. P. Atkinson, P. J. Bailey, K. Chisolm, W. Cockshott, R. Morrison. An approach to persistent programming. The Computer Journal, 26(4):360-365, 1983.

[ATK96] M. P. Atkinson, M. J. Jordan, L. Daynès, S. Spence. Design Issues for Persistent Java: a type-safe, object-oriented, orthogonally persistent system. Sun Microsystems Laboratories. February 1996.

[FER97] Paulo Ferreira, João Garcia. Fault Tolerance: Design and Exploratory Ideas. PerDiS Deliverable PDS-R-97-009.INESC, Lisboa, Portugal. 1997.

[GRA93] Jim Gray, Andreas Reuter, Transaction processing: concepts and techniques. Morgan Kaufman. California. 1993.

[JOR] Mick Jordan. Early Experiences with Persistent Java. Sun Microsystems Laboratories.

[SHA97] Marc Shapiro. PPF Architecture - general description. PerDiS technical report (http://www.perdis.esprit.ec.org/deliverables/docs/architecture/ppf-archi.html). INRIA. June 1997.

[TER95] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, Carl H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. *In* SIGOPS' 95. Colorado, USA. December 1995.

[WAN93] Randolph Y. Wang, Thomas E. Anderson. xFS: A Wide Area Mass Storage File System. In Fourth Workshop on Workstation Operating Systems, pp. 71-78. October 1993.