The PerDiS FS: A Transactional File System for a Distributed Persistent Store

João Garcia, Paulo Ferreira, Paulo Guedes

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

INESC / IST, Lisboa, Portugal

February 1998

Abstract

Companies cooperating in the framework of a virtual enterprise have increasing demands for systems on which to base applications for their particular environment: groups of workers on distant independent LANs. In this paper, we present a transactional file system for a distributed persistent store designed to support cooperative engineering applications. The system integrates techniques, such as optimistic consistency protocols and versioning, tailored to provide efficient sharing of data, between users (at LAN scale) and companies (at WAN scale).

1 Introduction

Current engineering projects are often developed by a virtual enterprise (VE), a temporary cooperation of independent companies. A VE comprises several geographically distant teams that work on separate local networks and on mostly distinct parts of a project. Each team consists of a group of workers that use workstations running engineering applications, e.g. CAD, which manipulate the VE's project data.

Traditionally, cooperative distributed applications are designed using remote invocation methodologies (function-shipping), such as CORBA [Sie96] or RMI [Wol96]. The performance of these client server applications degrades as objects become smaller and more numerous and as the number of clients increases. Furthermore, the wide-area invocation of remote servers raises latency and availability problems that are not addressed by function-shipping systems. Data-shipping designs copy data to the client machines allowing complex and long lasting manipulation of the data. In addition, this approach allows an optimisation of memory usage by allowing the creation of a shared address space among client machines.

We have designed a transactional file system, PerDiS File System (PFS), for a distributed persistent store [Sha97] that supports engineering applications in a VE environment. This work is being done in the context of the ESPRIT Persistent Distributed Store (PerDiS) project. The PerDiS design comprises interactions among computing nodes both at the local and wide area scales. This paper focuses on the local area architecture of the PerDiS transactional file system (PFS).

At a LAN scale, manipulation of the data is made on a set of client workstations running on a fast network with few reliable servers and a single protection domain. Advanced applications, e.g. design and development, resort frequently to long-lived access to data. To enable coherent and efficient sharing of data among such applications, optimistic transaction models are currently used [Fra97]. These protocols allow high degrees of concurrency and low abort rates in environments with low write contention such as the ones we are targeting.

Nevertheless, for the majority of the applications we are addressing, even a small abort rate is not acceptable for long-lived transactions. We use a versioning mechanism to reduce transaction aborts due to write contention. In addition, users want to perform tentative modifications and only later decide which one to submit to the server. Thus, one of the characteristics of VE applications is that different users may work on different copies, i.e. versions, of the same data. This involves the added complexity of managing a multi-version database. Yet, in cooperative applications, due to the parallelism it allows, this is perceived to be a good work methodology.

Committing divergent versions leads to the need of version reconciliation. Versions that do not satisfy traditional serialisation criteria, and cannot be merged automatically, have to be reconciled using user-driven application specific reconciliation tools. In our particular environment, companies participating in the VE conclude a specific task by electing a final version chosen (or composed) out of the version graph generated by their local processing, and disseminate it to other VE members.

The innovative aspect of our design is the integration of optimistic transactions with versioning mechanisms in order to satisfy the requirements of applications in a cooperative engineering environment. In the rest of this paper, we address relevant related work, give an overview of the PerDiS architecture and then focus on the PFS.

2 Related Work

Several data-shipping DBMS have extended standard pessimistic two-phase locking (2PL) protocols to allow a greater degree of concurrency and to improve memory utilisation [Lam91, Obj93, ONT94]. In particular, optimistic locking protocols allow intense concurrency under low write contention.

A client/server DBMS with inter-transactional caching at the clients requires a remote update mechanism to maintain cache coherence by updating the clients' caches when other clients commit transactions using the same data. This can be achieved by using invalidation of stale cache data, propagation of the new committed data or by a dynamic choice between these techniques [Fra97]. It has been shown that, in cooperative environments with low write contention, protocols that use invalidation perform best because they are less sensitive to workload patterns. It is worth noting that, in our case, notifications never imply an abort given the support for versioning. Invalidation messages may be used to notify applications of other conflicting users.

Regarding versioning, a variety of schemes have been implemented for DBMS [Lam91, Obj93, ONT93]. Most versioning schemes are based on the concept of a configuration, i.e. a graph of objects that constitutes a versioning unit. The simplest versioning mechanism is linear versioning that designates the general approach of tagging successively committed versions of a set of data, with version numbers or timestamps, so that applications can identify the older versions they wish to rollback to.

More complex mechanisms include branching versioning [Lam91, Obj93] and Objectivity's move-copy-drop techniques [Obj93]. Branching mechanisms allow users to create divergent versions of data. This may derive from the need of committing transactions that conflict according to standard serialisation criteria or merely from the wish of maintaining different copies of the data.

More specific mechanisms to modify configurations are used in Objectivity [Obj93]. This system has a particular set of operations, move-copy-drop, to commit objects to the database. The copy operation triggers a full copy of the graph containing the newly versioned object; move replaces the old version of the object in the configuration while maintaining the previous version; and drop erases the previous version of the committed object.

3 PerDiS Architecture

PerDiS is an infrastructure for programming distributed, type-safe applications that transactionally access persistent data (Fig.1). PerDiS is designed for a virtual enterprise environment where machines on several LANs cooperate temporarily over a wide-area network.

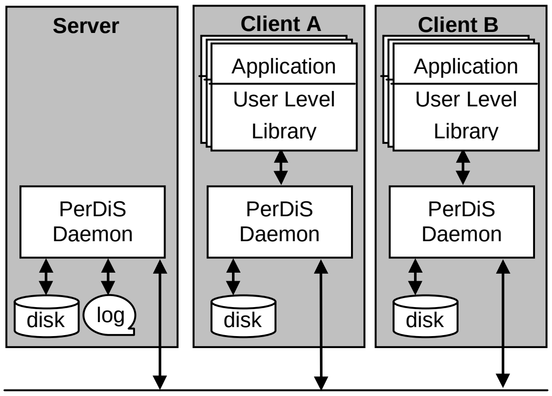

Figure 1 PerDiS System Architecture

It allows applications to manipulate files containing graphs of objects which we call clusters. Clusters are stored persistently at servers' stable storage according to the model of persistence by reachability [Atk83]. This persistence model guarantees that all objects that are reachable, directly or indirectly, from a named root object will be made persistent. PerDiS applications are linked to a user level library (ULL). This ULL communicates with a PerDiS Daemon that manages locking and handles all persistent and fault tolerant data.

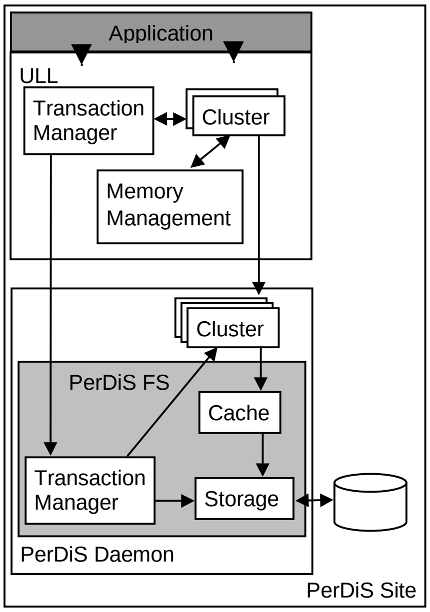

Figure 2 Modules in PerDiS

The PerDiS File System (PFS) provides the transactional, caching and storage support for PerDiS (Fig.2). Other modules in PerDiS handle object clusters (swizzling, reference management, clustering) and handle type and memory management.

Besides the local area activity described in section 4, PerDiS also operates on a wide-area scale. At this dimension, company servers interact to provide coarse granularity data, i.e. files, to other cooperating LANs and to return newly generated versions of the data to the original sites. Typically, each member of the VE is responsible for the data it creates. For example, in a building VE, the architecture company owns the master copy of the architecture project: it may selectively distribute sub-sets of its data to other VE members and it is the ultimate responsible for accepting changes to the architecture project. These wide-area exchanges over untrusted networks are encrypted and file access is authenticated.

4 PerDiS File System

The PFS is a distributed transactional file system. On a local scale, it operates within a company LAN's protection domain. Servers manage transactions and stable storage. Client machines run applications that access the persistent store and manage a page-based cache of persistent data. The PerDiS File System stores object clusters in files on the servers' disks. These clusters are referenced using URLs.

Clients request file blocks and memory map them; servers keep track of the pages issued to each client. At the clients, pages are cached and may be shared for reading. Write accesses to memory pages are performed on copies exclusive to each application.

Cache coherency among clients is maintained using the entry consistency protocol, which takes advantage of the low contention among clients and facilitates the probable sequential access by the client holding the lock [Fra97]. Besides invalidating other client caches, entry consistency invalidation messages may also be used as notifications to applications that others are using the same data.

The use of entry consistency raises some fault tolerance issues. Recovering this protocol's data after a node failure, especially probable owner pointers and locks possibly lost at the failed node, is not a trivial problem, which we will address in the near future.

4.1 Transaction Management

Each PerDiS application runs a single transaction at a time. As already mentioned, pessimistic transactions have a limited usability in cooperative engineering applications. Thus, optimistic transactions are needed in order to allow users to combine data sharing and long lived transactions. Hence, we use an optimistic protocol with notifications. In the PFS, to avoid aborting long transactions, upon receiving notifications, PerDiS daemons only invalidate pages that are currently not involved in any transactions. Invalidation notifications for data being accessed are forwarded to the application level for user information.

Distributed transactions are committed using a non-blocking two-phase-commit protocol. When a client commits a transaction it is written onto the local disk. We rely on the local disk to stably maintain the transaction's updates during its commit protocol. Once the updates are written on the local disk, the client application may continue executing.

Guaranteeing that long lived optimistic transactions don't abort due to data contention, requires, in the first place, always providing a consistent view of the data to all active transactions. A transaction that begins at a given point in time will have to consistently receive the data that was store in the servers' disks at the time it begun, until it ends. Stretched to the limit this requirement would force us to maintain all the versions that have ever existed of all cluster files, because there might be somewhere in the system a node with a long-lived transaction that might need data from a long time ago. However, due to disk space restrictions, we must differentiate between old data that might still be requested and old data that does not belong in any consistent view of the cluster files being provided to active transactions. To this effect, we are going to develop a client register service, that will ensure, to clients that register themselves, that clusters that were valid when their active transaction began, will be maintained until they commit the transaction or unregister themselves as active clients.

During the commit protocol, since we already guaranteed that consistent data was provided to the committing transaction, servers that provided data, that was not modified, need not be contacted. Our versioning support guarantees that applications will not lose work done within transactions. This may be a traditional database commit where, after transactions are committed, there is a single image of the database. As a last resort, in the cases where optimistic transactions have conflicts and their serialisation is not possible, we extend the semantics of the commit operation by committing both transactions to the servers' disk as alternative versions of the same data. In the following section, we discuss the management of versions.

4.2 Versioning and Checkpoints

Our system guarantees that optimistic transactions aren't aborted due to write contention. Conflicting transactions are committed using a branching versioning mechanism. Furthermore, if an application, during a transaction wants to checkpoint its persistent data without submitting it to the whole system, it can save it onto the local disk. This checkpoint, invisible to other users, can later be recovered and work resumed at that stage.

Transactions performed by a local group of users will result, due to contention or divergence of concepts, in a graph of versions. In the virtual enterprise example, it is the responsibility of the company that initially created the cluster to elect one of the committed versions of the store as the final version to be publicised to other participating networks. The low write contention among concurrent design application helps keep the amount of branching in a cluster's version graph to a minimum.

Many versions, especially those generated due to write contention, cannot be merged automatically. This leads to the need of programmer/user aided tools based on semi-automatic reconciliation algorithms, e.g. anti-entropy [Gol92, Pet97, Ter95], to reconcile versions of the store. Our design provides support for this functionality through APIs for editing, labelling and navigating graphs of versions.

4.3 Global Memory Management

Another interesting aspect of operating on a local network with a single protection domain is the possibility of optimising the occupation of the network's aggregate memory and data access speed by including cooperative caching techniques in a transactional environment.

The usual techniques involved are moving pages to other machines' memory ("dropping the page") instead of writing them to disk and the consequent forwarding of any messages regarding those pages [Dah94, Dan92, Fra92]. However, in our system, we have to consider some new aspects such as transactional consistency and versioning when making use of other machines' memory.

We have chosen to use memory pages as the unit of cache coherence because this allows efficient integration with the virtual memory system and consequently write access detection using the page fault mechanism. Pages are cached between transactions until they are invalidated when they become stale, forwarded ("dropped") to other clients' memory, or, as a last resort, evicted to disk.

A consequence of this approach is that some clients, instead of getting data from a server, will have their request forwarded to some other site which has the data needed. Furthermore, the use of versioning may lead to the existence of alternative copies of the same memory page in the clients' caches.

5 Conclusions & Future Work

This paper described the motivation for the PerDiS transactional file system and some of its most relevant aspects. The contribution of this work is the integration of transactions, versioning and global memory management for VE applications.

Cooperative applications have special requirements such as the support for long-lived transactions that should never abort. Alternatively, new, possibly divergent, versions of data are created thus requiring reconciliation. Meanwhile, the graph of these versions has to be managed (edited, labelled and pruned). We discussed how these mechanisms can be introduced in a client/server distributed store with a shared address space cache among clients.

Even though this is still work in progress, there is already a preliminary working prototype available on the Internet (http://www.perdis.esprit.ec.org) that runs all several platforms (Sun Solaris, Linux, Windows NT) and with which the end-user partners of the PerDiS project are currently experimenting.

There are several developments that are essential to help mature PerDiS and the PFS.

Fault tolerant locking is fundamental in order to have a serious platform, since it is unacceptable that a node failure may block the whole system.

Versioning as a means to resolve transaction conflict is a simple but powerful mechanism that enables many models and policies for cooperative work that need to be explored.

Acknowledgements

The authors wish to thank all the partners in the PerDiS project (INRIA SOR, INRIA SIRAC, IEZ, CSTB, QMW) for their valuable participation in designing and implementing PerDiS and for their input in various discussions that have been influencing the design of PFS.

Bibliography

[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.

[Ale97] Albert D. Alexandrov, Maximilian Ibel, Klaus E. Schauser, and Chris J. Scheiman. Extending the Operating System at the User Level: the UFO Global File System. *In* USENIX Winter, Anaheim, California (USA), January 1997.

[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.

[Bar91] Naser S. Barghouti and Gail E. Kaiser. Concurrency control in advanced database applications. Computing Surveys, 23(3):269-317, September 1991.

[Car94] Michael J. Carey, David J. DeWitt, Michael J. Franklin, Nancy E. Hall, Mark L. Auliffe, Jeffrey F. Naughton, Daniel T. Schuh, Marvin H. Solomon, C. K. Tan, Odysseas G. Tsatalos, Seth J. White, Michael J. Zwilling. Shoring up persistent applications. *In* Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 383-394, Minneapolis MN, USA, May 1994.

[Dah94] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. *In* Proc. Symp. on Operating Systems Design and Implementation, pages 267-280, Monterey CA (USA), November 1994.

[Dan92] A. Dan, P. Yu. Performance Analysis of Coherency Control Policies through Lock Retention. Proc. ACM SIGMOD Int. Conf. On Management of Data, San Diego, CA. June 1992.

[Deu91] O. Deux et al. The 0_2 system. Communications of the ACM, 34(10), October 1991.

[Fra92] Michael J. Franklin, Michael J,. Carey, Miron Livny. Global Memory Management in Client-Server DBMS Architectures. Proc. Of the 18th

Int. Conf. On VLDB, Vancouver, Canada. August 1992.

[Fra97] Michael J. Franklin, Michael J. Carey, Miron Livny. Transactional Client-Server Cache Consistency: Alternatives and Performance. ACM Transactions on Database Systems, Vol.22, No.3, pp.315-363. September 1997.

[Gol92] R. A. Golding. A weak-consistency architecture for distributed information services. Computing Systems 5(4):379-405, Fall 1992.

[Lam91] Charles Lamb, Gordon Landis, Jack Orenstein, Dan Weinreb. The ObjectStore Database System. Comm ACM 34(10), pp. 50-63. October 1991.

[Lis92] Barbara Liskov, Mark Day, and Liuba Shrira. Distributed object management in Thor. *In* Proc. Int. Workshop on Distributed Object Management, pages 1-15, Edmonton (Canada), August 1992.

[Obj93] Objectivity, Inc.. Objectivity Tech. Overview. 1993.

[ONT94] ONTOS Inc.. Introduction to ONTOS DB 3.0, ONT-30-SUN-IODB-1.0. April 1994.

[Pet97] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, Alan J. Demers. Flexible Update Propagation for Weakly Consistent Replication. *In* Proceedings of the 16th ACM Symposium on Operating Systems Principles. October 1997.

[Sat89] M. Satyanarayanan. A survey of distributed file systems. Technical Report CMU-CS-89-116, Department of Computer Science, Carnegie-Mellon University, Pittsburgh PA (USA), February 1989.

[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.

[Sie96] Jon Siegel. CORBA Fundamentals and Programming. John Wiley & Sons, Inc., 1996.

[Sun89] Sun Microsystems, Inc. NFS: Network file system protocol specification. RFC 1094, Network Information Center, SRI International, March 1989.

[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.

[Vah96] Amin M. Vahdat, Paul C. Eastham, and Thomas E. Andersom. WebFS: A global cache coherent file system. Technical report, University of California - Berkeley (Computer Science Division), Berkeley, CA 94720, December 1996. http://now.cs.berkeley.edu/WebOS/publications.

[Wol96] Ann Wollrath, Roger Riggs, Jim Waldo. A distributed object model for the java system. *In* Conference on Object-Oriented Technologies. Toronto Ontario (Canada). Usenix. 1996.