Amino: Extending ACID Semantics to the File System

Charles P. Wright, Richard Spillane, Gopalan Sivathanu, and Erez Zadok

An organization's data is often its most valuable asset, but today's file systems provide few facilities to ensure its safety. Databases, on the other hand, have long provided transactions. Transactions are useful because they provide atomicity, consistency, isolation, and durability (ACID). Many applications could make use of these semantics, but databases have a wide variety of non-standard interfaces. As a result, applications like mail servers currently perform elaborate error handling to ensure atomicity and consistency, because it is easier and more portable than using a DBMS. Most editors write changes to a temporary file, and then rename the temporary file over the original to replace the content atomically, but there is no standard method to atomically update two related files (e.g., `/etc/shadow` and `/etc/passwd`). A transaction-oriented programming model provides three key benefits: (1) complex error-handling code is avoided, because failed operations can simply be aborted; (2) concurrent accesses behave as if they were serialized (thereby preventing time-of-check-time-of-use security vulnerabilities); and (3) once a transaction is committed, it will not be lost due to software or hardware failures.

We believe that file systems should export transactions as a first-class service. In this way, applications can continue to use the simple, flexible, and pervasive POSIX API to access their data, but do so in a transactionally protected environment. Furthermore, other applications, which may not need such transactional protections can still access the data, without any changes. To support transactions properly, both file systems and the OS itself must handle transactions. File systems must log redo and undo information, so that transactions can be applied or aborted. The OS also needs to support transactions: traditional caches (e.g., the page cache or directory-name-lookup cache) return data to user space applications without consulting the file system. This creates several problems for transactional file systems. If the database management system does not mitigate all accesses, then the ACID properties cannot be guaranteed. If a transaction was aborted, then the OS caches will contain stale data—violating atomicity. If the database is not consulted before accessing an object, then it can't perform proper locking—violating isolation. This dictates that an OS which supports transactions must have *database-managed caches*.

On further examination, database-managed caches are only a first step. When accesses are mitigated through the database's caches, the file system is consistent and the user-level application can interact with it transactionally, but other OS components do not share this consistent state. For example, if a file is created by opening it with the `O_CREAT` option, then a new file descriptor is created and inserted into the OS's process control block (PCB). If the transaction is later aborted, then the PCB will point to a file descriptor for a non-existent file. Clearly, these issues demand support for transactions in the OS proper. This means that it is useful for the OS to extend transactions as far as possible, including to the application so that its data structures can be kept consistent with the file system.

Status. We have designed a prototype file system that exports ACID transactions to user-level applications, while preserving the ubiquitous and convenient POSIX interface. In our prototype ACID file system, called *Amino*, unmodified applications operate without any changes. For these unmodified applications, each system call is transaction protected. Using Amino, application developers can protect an arbitrary sequence of system calls using transactions by inserting simple BEGIN, COMMIT, and ABORT calls. Other transactional file systems such as QuickSilver [3] only run on specialized OSes designed from the ground up for transactions, or change the file access API (e.g, QuickSilver cannot randomly write to a file and WinFS [1] changes the file access API to one that uses items instead of files), preventing existing applications from accessing the data.

We built our prototype on top of the Berkeley Database (BDB), a time-tested and reliable embedded database. Our prototype intercepts system calls at the ABI-level via a `ptrace` monitor to provide OS services to any application from user-level. Because we intercept operations at the ABI, any existing application can be run through our monitor. More importantly, we are operating above the OS, so that we can use database-managed caches, and transactionally protect all of the relevant file system structures (including PCBs). To protect these structures, we developed a recoverable virtual memory (RVM) [2] system for our monitor and user applications. Our RVM system improves on previous systems by allowing nested transactions and transparently logging updates via page-protection.

Our performance evaluation shows that ACID semantics can be added to applications with acceptable overheads. When Amino adds atomicity, consistency, and isolation functionality to an application, it has an overhead of 15.4% over Ext3 due to overhead of the monitoring infrastructure. When atomicity, consistency, isolation and durability are provided, it is up to 26.6% faster than Ext3, because BDB's balance tree structure has improved locality and is optimized for durable performance.

Our ongoing research can be divided into three main categories. First, we are focusing on improving performance, particularly under data-intensive workloads. Second, we are investigating the relationship between the four database isolation levels and performance. Third, we are investigating improved `ptrace` primitives to reduce the overhead of our monitoring infrastructure.

References

[1] Microsoft Corporation. Microsoft MSDN WinFS Documentation. `http://msdn.microsoft.com/data/winfs/`, October 2004.

[2] M. Satyanarayanan, H. H. Mashburn, P. Kumar, D. C. Steere, and J. J. Kistler. Lightweight recoverable virtual memory. *ACM Transactions on Computer Systems*, 12(1):33–57, 1994.

[3] F. Schmuck and J. Wylie. Experience with transactions in QuickSilver. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91)*, pages 239–253, Pacific Grove, CA, October 1991. ACM Press.