TxFS: Leveraging File-System Crash Consistency to Provide ACID Transactions

To recover to a correct state after a crash, most of applications effort to ensure crash consistency. However, POSIX lacks an efficient atomic update to multiple files. So author argues that Transactional API enables the file system to provide a number of optimizations such as Eliminating temporary durable files, Group commit, Eliminating redundant IO within transactions, Consolidating IO across transactions, and Separating ordering from durability. TxFS's Contribution is to providing transactional file systems with high performance.

A simple API was one of the key goals of TxFS. TxFS provides developers with only three system calls for transaction begin, commit, and abort. TxFS builds upon the ext4 file system's journal which is write-ahead-logging. So TxFS takes A,C and D properties with file system's journaling. For TxFS, providing isolation for concurrent execution between multiple transactions is the main challenge. To avoid false conflict, TxFS makes a transaction private copies of all kernel data structures modified during the transaction. TxFS divides file system functions into two parts. The First is operation that modify locally visible state executed on private data structures copies. Second is the operation modifying global state delayed until commit time. TxFS detect conflict with different mechanisms for page and inode/dentry. Because kernel provides interface for detection page access by using page caching. So each transactions can find that if there are other transaction's access when they read or write to target page. This is the eager conflict detection. However, there are no standard interface to detect whether conflicts are existed on inode and dentries or not. To resolve this problem, TxFS applies lazy conflict detection which is done at commit point by using timestamp.