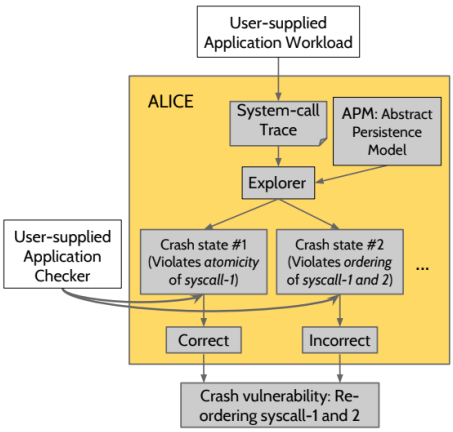
All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications, OSDI 2014

This paper found that applications use complex update protocols to persist state, and that the correctness of these protocols is highly dependent on subtle behaviors of the underlying file system, persistence properties.

By using BOB(Block Order Breaker), the tool they made, they shows persistence properties vary widely among file systems (even with different configurations of the same file system). This paper said it is risky to assume any particular property will be supported by all file systems, on developing applications.

They build ALICE(Application-Level Intelligent Crash Explorer), a framework that analyzes application-level protocols and detects crash vulnerabilities. And by testing multiple applications, they show how application-level consistency is dangerously dependent upon file system persistence properties. Using ALICE they analyze 11 applications, find 60 vulnerabilities, some of which result in severe consequences like corruption or data loss.



Pros

It shows that consistency of applications is dependent on file system persistence properties.

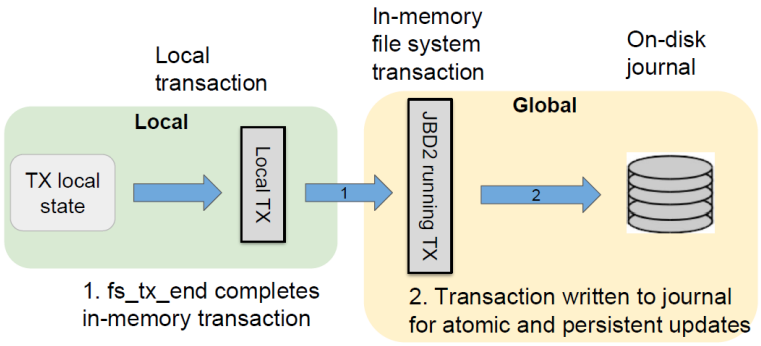
Cons & Ideas

Not easy to use ALICE, user should write workloads and checkers. → Make workloads building procedure automatic.

TxFS: Leveraging File-System Crash Consistency to Provide ACID Transactions, ATC 2018

TxFS reuse file-system journal for atomicity, consistency, durability

For transactional behavior, TxFS splits transaction into two parts: local and global transaction



For isolation, TxFS makes copies of all kernel data structures modified during local transaction. File-system related system calls inside a transaction operate on these private copies, allowing transactions to read their own writes.

In local transaction, to avoid confilict on global data structure, TxFS make copies and update inodes, dentrie and data pages in private(local). On **updating inodes and dentries**, they use copy-on-read mechanism and serve lazy conflict detection by checking time stamp on commit. On **updating data pages**, they use copy on write mechanism for eager conflict detection, they use fine-grained page locks for high scalability. In case of abort, which means conflict detected on local transactons, private copies are discarded.

On global transaction, these private copies are carefully applied to the global state of the file system in an atomic fashion.

Pros

build a transactional file system with low complexity by reusing file-system journaling. and Increases performance significantly for a number of workloads.

Cons & Idea

If storage device use multi IO submit queues and reorder IOs, are they tolerant with it?

Since local writes are visible only to dedicated thread, cannot consider thread dedicated to multiple threads.

It would be better to show example of recover procedure.