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

This paper shows how application-level consistency is dangerously dependent upon file system *persistence properties*, i.e., how file systems persist system calls. The *persistence properties* of modern file systems determine which possible post-crash file system states are possible for a given file system; different file systems provide subtly different guarantees, making the challenge of building correct application protocols atop such systems more vexing. To study persistence properties, this paper introduces a tool, known as the Block order Breaker (BOB), to empirically find cases where various persistence properties do not hold for a given file system. Also, they build a framework named ALICE that analyzes application update protocols and finds crash vulnerabilities. Using ALICE, they analyze many widely-used systems and find a total 60 vulnerabilities. In addition, they say ALICE can be used to evaluate the effect of new file-system designs on application-level consistency.

This paper has strengths that it is been interesting to start by revealing the fact that many applications today cannot guarantee crash consistency on various modern file systems even between configurations of the same file system. And through many analyzes and many examples, it has helped to understand the problems to be addressed in the paper and the tools to find these problems.

However, As noted at ALICE's limitations, the weaknesses are that ALICE cannot detect all vulnerabilities and does not handle file attributes. Also, it is too early to say that ALICE can be used to evaluate the effect of new file system through a single experiment.

I hope that there are more examples to fix the vulnerabilities discovered through ALICE to ensure crash consistency.

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

Modern applications need to ensure that their data is not corrupted or lost in the event of a crash. Unfortunately, existing techniques for crash consistency result in complex protocols and subtle bugs. In this paper, they introduce a novel approach to building a transactional file system called TxFS. They use the atomicity and durability provided by journal transactions and leverage it to build ACID transactions available to user space transactions. TxFS can obtain atomicity, consistency, and durability by using well-tested journal code in file system. And TxFS provides isolation by making private copies of all data that is read or written, and updating global data during commit. Plus, TxFS makes concurrent transactions efficient by collecting logical updates to global structures, and applying the updates at commit time. In result, TxFS can support ACID transactions with a simple implementation.

This paper has strengths that it has released code that is available, has proven TxFS through

various experiments result, and that TxFS has shown better results in performance and crash semantic as easy fixes in real applications.

The drawbacks of this approach are the complexity and incompleteness of the interface, and the a significant implementation complexity. According to the paper [File Systems Unfit as Distributed Storage Backends: Lessons from 10 Years of Ceph Evolution] (SOSP 2019), its implementation requires non-trivial amount of change to the Linux kernel.

In the database experiments, It would be better that the results are not only in relational databse (SQLite) but also in key-value stores (e.g., LevelDB and RocksDB).