Aiman Priester

P16: PFault, A General Framework for Analyzing the Reliability of High-Performance Parallel File Systems

Summary

Authors of this paper introduced a framework to analyze failure handling of Parallel File Systems (PFS), This is motivated by a lack of an effective analysis methodology, which could have prevented incidences of data-loss after a power fault as seen in the High-Performance Computing Center in Texas Tech. The paper aims to uncover issues present in PFSs, focusing on Lustre, a widely used open-source PFS. By emulating whole device failures (a-devFail), global inconsistencies (b-Inconsist) and network partitioning (c-Network), PFault is found to expose Lustre's shortcomings in failure handling. PFault was found to also expose LFSCK, Lustre's verification and repair tool, to fix and perform with consistent results under different faults. The authors went ahead and proposed a patch for LFSCK, which aims to fix resource leak error reporting that went unreported by LFSCK. With this, authors hope that PFault can be used alongside other analysis tools to bring errors into the light to avoid incidents that happened to HPCC occur.

Strengths

Unlike other distributed file systems, PFSs are designed around the ability to concurrently access a file. How a PFS reacts to real-world issues such as power faults vastly impacts the user, in this case, academia at large. With this in mind, the PFault's focus on I/O operations is a big upside. I/O operations must work to uphold PFSs concurrent access property.

Early in the paper, it is noted that almost every layer of the storage stack has been found with reliability vulnerabilities. PFault does a great job of isolating the PFS from the rest of the storage stack while maintaining a consistent file system. This isolation will ease the debug process for future users, as it removes the possibility of an unexpected error that occurred in a different layer.

Finally, PFault clears reliability assumptions in LFSCK in its ability to detect and handle issues. LFSCK is found to be unable to fix or report memory leak issues, to which the authors went ahead and proposed a patch, LeakCK. This patch allows the reporting of potential memory leaks (Read: Unreachable Files) to the user.

Weaknesses

While reading this paper, I was interested in the reasoning behind using two versions of Lustre. Versions 2.8 and 2.10 were checked using PFault. Curiosity has led me to find that 2.8 was released in February 2016 and 2.10 was released in July 2017. The paper did not address the reason behind the usage of those two versions. Also, given the relatively fast development of Lustre, it leaves to be answered as to why 2.10 was tested in the way it was. Version 2.8 was tested against 2.10 and it is found to have improvements. Would this insinuate that improvements were not expected? In my opinion, a more feasible approach would be to specifically test fixes documented in the Lustre changelog against its release. I suppose one could argue that testing two different versions could verify the effectiveness of PFault, but addressing this in the paper would clear a few assumptions.

Problems / Future Work

It is said that LeakCK reports “unreachable” objects as a potential leak. Then, it optionally moves them to a lost + found directory. Since this process is optional, who determines this action. Is it optional in terms of user implementation, or does LeakCK choose in its own way to perform this action?

While it is a sound assumption that LeakCK might be applicable to other PFSs, it would be heavily beneficial to further this study and emulate this solution to other PFSs. This is to have PFault to be used as a general framework that analyses and potentially patches minor issues, as it aims to be. (OrangeFS maybe?)

Discussion

Since PFault modularizes checking to only the PFS layer, what would the caveat or limitation be to a team that wishes to implement a more “full-stack” analysis tool, combining multiple tools for each layer of the storage stack?