A Centralized Failure Handler for File Systems

Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau
Computer Sciences Department, University of Wisconsin - Madison

1 Motivation

Failure handling in file system is broken. Commodity file systems are built under the assumption that disks fail in a fail-stop manner. However, in reality, portions of a disk can fail by latent sector errors or block corruptions. The code that handles disk failures is distributed throughout the file system along with the I/O calls. This diffusion of failure handling results in several problems:

• Illogically inconsistent failure handling policies: File systems use different failure handling techniques even under similar fault scenarios [3].

• Tangled policies and mechanisms: It becomes harder to separate failure policies from detection and recovery mechanisms.

• Diffusion of bugs: There are several bugs in the file system failure handling code. Block I/O failures are even ignored sometimes. Since the failure handling is repeated at several places, it is hard to fix them all [2, 3].

In this work, we propose to design, implement, and evaluate a centralized failure handler that detects and recovers from I/O failures with well-defined failure policies that are easier to understand and debug.

2 Centralized Failure Handling

A centralized failure handler is a file system subcomponent, similar to the buffer cache manager or the journaling layer. It controls all the I/O initiation and completion. It detects block read-write failures and corruptions, and invokes the specified recovery policy.

A centralized failure handler gives several benefits. By handling failures in one place, inconsistent policies can be eliminated. New functions that manipulate I/Os can be added relatively easily since the developer is relieved of the burden of writing a failure handler for each such function. The failure handler can be viewed as a way to separate failure policies from detection and recovery mechanisms [1]. Policy decisions such as whether to protect blocks using replicas or parity can be separated from their implementation details. Finally, placing all the failure handling mechanisms at one place makes them easier to debug and fix.

A centralized failure handler can be used to support fine grained failure policies. File system can specify different failure policies for different block types and I/O contexts. Applications can also specify failure policies appropriate to their needs. For example, an application might want to replicate an important directory while requesting no redundancy for a temporary file.

There are three main issues in designing and building a centralized failure handler. They are:

Architecture: The failure handler interacts with the core file system that manages the data and metadata, and with other components of the file system such as the cache and journal. The failure handler contains two subcomponents: a file system aware part that deals with different block types, transaction semantics, and I/O contexts; and a generic part that implements the detection and recovery mechanisms, which can be used commonly across several file systems.

Information: I/O requests can be issued under different contexts for different block types, and failure policies can change accordingly. Therefore, the failure handler needs semantic information about block I/Os to select the appropriate failure policies. The failure handler must be aware of semantic information such as the logical boundaries of a file, contents of a transaction, and block to file mapping. The file system must provide appropriate interfaces to get such information. The failure handler maintains a table that maps the block types and context information to their appropriate failure policies.

Machinery: Implementing a centralized failure handler requires re-architecting parts of the file system. New machinery must be added to direct the I/O calls to go through the failure handler. I/O paths are time critical. Completion of blocks I/Os are notified in the interrupt context. Complex failure handling mechanisms cannot be executed under the interrupt context. The failure handler contains additional machinery to separate I/O completion path from failure handling.

3 Summary

Failure handling is diffused in file systems. This results in inconsistent policies, tangled policies and mechanisms, and diffusion of bugs. We are currently working on building a centralized failure handler as a way to support logically consistent failure policies and separate policies from mechanisms. It requires file system specific semantic information to support fine-grained failure policies.

References

[1] R. Levin, E. Cohen, W. Corwin, F. J. Pollack, and W. Wulf. Policy/mechanism separation in Hydra. In *Proceedings of the 5th ACM Symposium on Operating Systems Principles (SOSP '75)*.

[2] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Model-Based Failure Analysis of Journaling File Systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2005)*, pages 802–811.

[3] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. IRON File Systems. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*.