Physical Disentanglement in a Container-Based File System

Lanyue Lu, Yupu Zhang, Thanh Do, Samer Al-Kiswany
Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

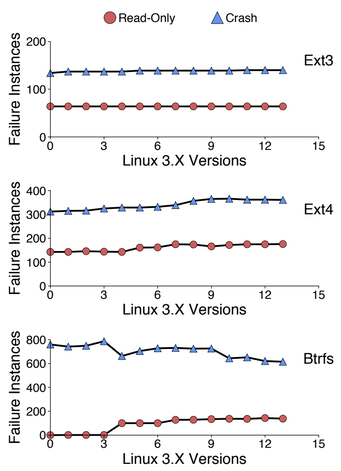
Motivation

File systems lack isolation

Various techniques (virtual machines, Linux containers, BSD jail, Solaris Zones) are based on namespace isolation.
However, namespace isolation is not enough.

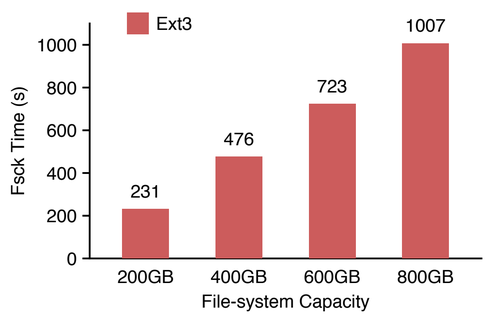
Global failures

Read-only and crash are common in file systems.
A fault in shared structures affect multiple unrelated files, directories and even namespaces.



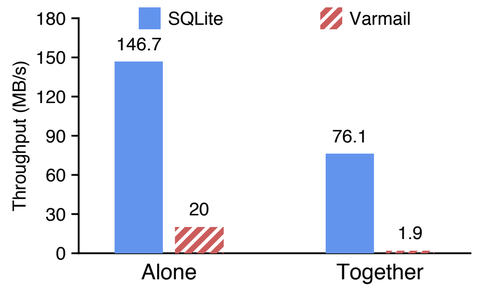
Slow recovery

File system checkers scan the whole file system regardless of the faulty range.
Current file system checkers are not scalable.



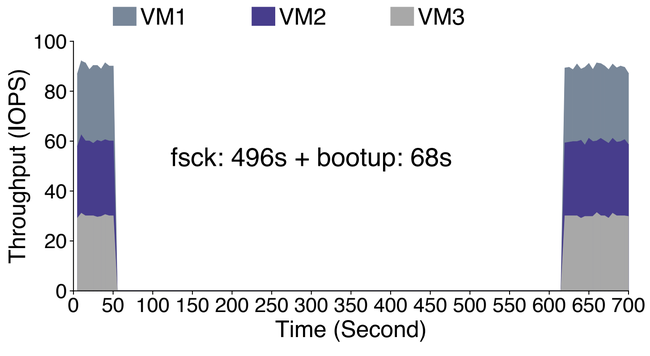
Bundled performance

Modern file systems use a shared transaction for updates from different files.
The performance and consistency of independent applications is bundled.



Virtual Machines

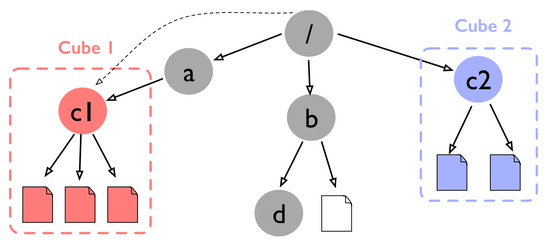
Many VMs run on a shared hypervisor file system.
If a fault in one of VM's disk image makes the file system read-only, then all VMs will be affected.



IceFS

The Cube abstraction

A group of files are both logically and physically isolated in the file system. Each cube is independent at the file system level.
Interface: create a cube, set attributes, add files, delete files, remove a cube.

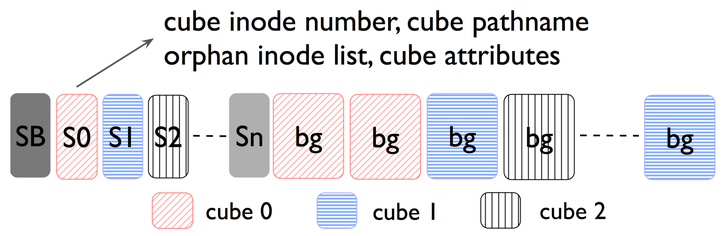


Disentangled data structures

Key data structures within modern file systems must be disentangled to support cube abstraction.
Three principles of disentangled data structures:

➡Physical resource isolation

Cubes must not share physical resources, such as shared metadata, disk blocks and memory buffers.

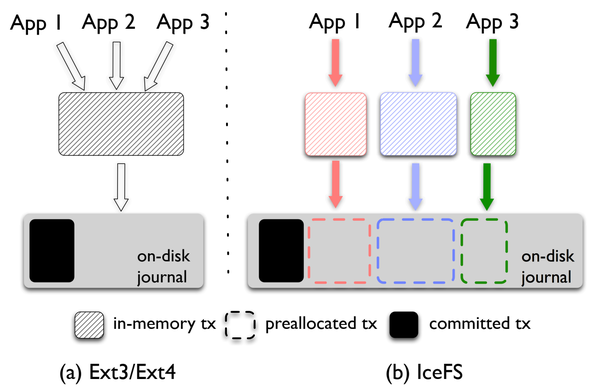


➡Access independence

Each cube does not contain references to other cubes. Directory indirection is use to provide independent access for cubes.

➡Transaction splitting

Each cube maintains its own separate transactions. Different cubes commit to the journal in parallel with journal space reservation.



Benefits of disentanglement

Three major benefits of a disentangled file system:

➡Localized reactions to failures

The failure of a cube can be detected and handled locally. A faulty cube can be marked as readonly or crashed without affecting other healthy cubes.

➡Localized recovery

A cube can be viewed as a basic checking unit instead of the whole file system. Both offline and online checking are feasible now in IceFS.

➡Specialized journaling

Parallelized journaling framework in IceFS enables isolated performance for cubes. Each cube also can have customized journaling modes for flexibility of consistency and performance tradeoffs.

Evaluation

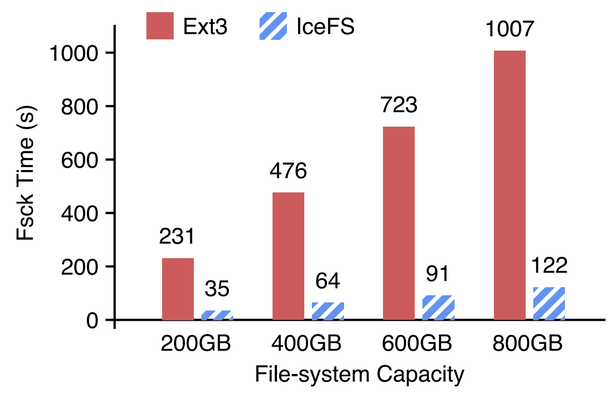
Overall performance

IceFS is implemented in Linux 3.5, based on Ext3 and JBD. The total lines of code is about 7800 (6200 in Ext3/JBD, 960 in VFS and 700 in e2fsprogs)

For both micro (sequential read/write) and macro (Fileserver, Varmail, Webaserver) benchmarks, IceFS incurs little overhead compared with Ext3.

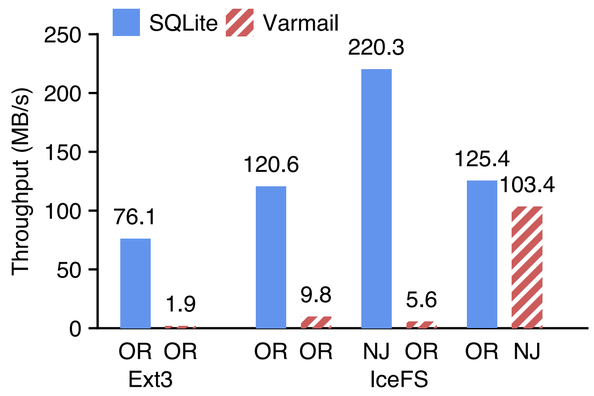
Fast recovery

IceFS only performs checking on the faulted cube. For different file system sizes, this figure shows the file system recovery time (fsck) of Ext3 and IceFS for one faulty cube (total 20 cubes).



Specialized journaling

We show that a disentangled journal enables high performance for competing applications and different consistency modes in a shared file system. Both SQLite and Varmail run simultaneously on both Ext3 and IceFS, with a SSD as the storage device.



Virtual machines

We demonstrate that IceFS can isolate file system failures in a virtualized environment and improve the availability of the system significantly by reducing the system recovery time. Different behaviors of both offline and online recovery are also shown.