Physical Disentanglement in a Container-Based File System OSDI'2014

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

University of Wisconsin, Madison

夏飞
2014.11.27

Outline

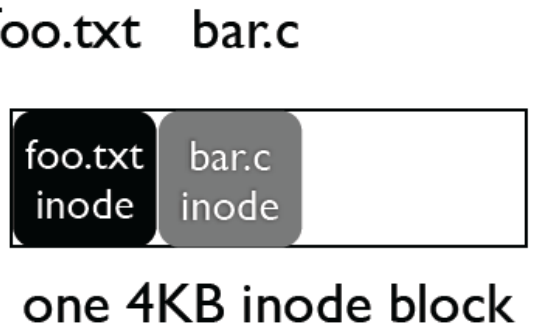
- Introduction
- Motivation
- File System Disentanglement
- IceFS
- Evaluation
- Related Work
- Conclusion

Introduction

- Isolation is important
  - Reliability
  - Performance
- Various isolation techniques for:
  - CPU
  - Memory
  - I/O bandwidth
  - Device driver
  - Shared storage cache
- What about file system?

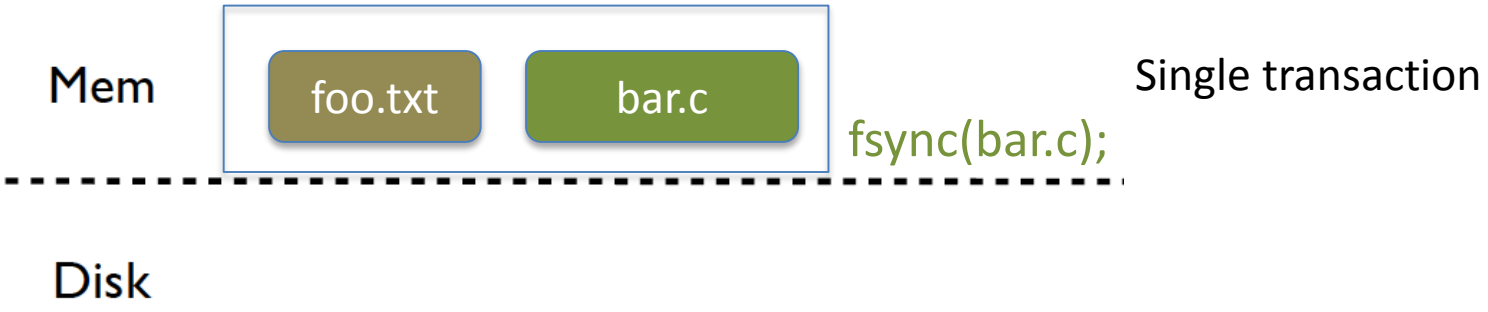
Local File System Lacks Isolation

- Existing abstraction provides logical isolation
  - File, Directory, Namespace
- However, local FS is physical entangled
  - Data structure
  - transaction
- Example: Metadata entanglement
  - Multiple files share one inode block
  - Shared data structure:
    - bitmap, directory
- Problem
  - Faults in shared structure result in shared failure and recovery.



Local File System Lacks Isolation

- Example: Transaction entanglement
  - Ext3 uses JBD to record journal
  - JBD groups multiple updates into a single global transaction to achieve better throughput.



- Problem
  - Transaction entanglement results in entangled performance.

- Entanglement Problems
  - Global failure

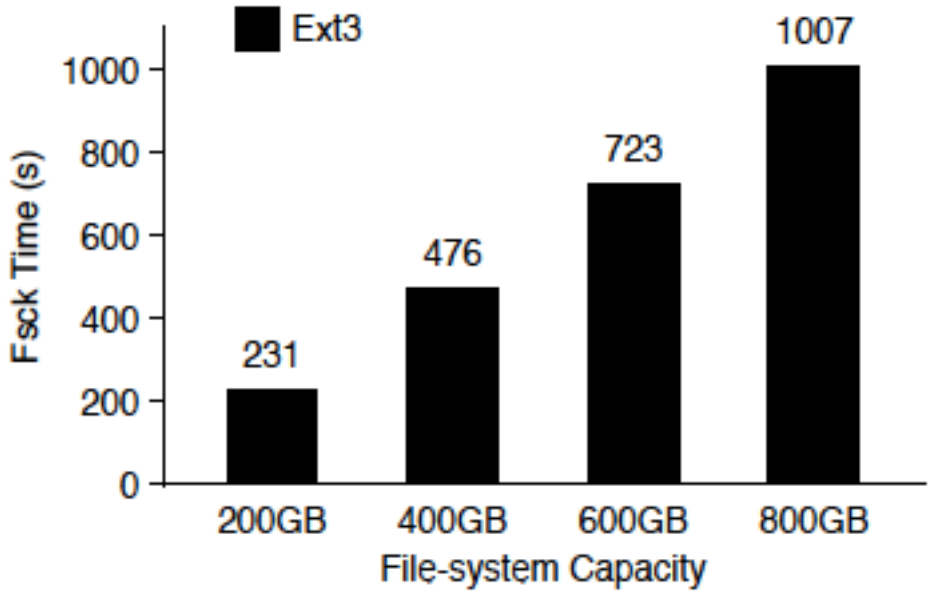
Global Failures	Ext3	Ext4	Btrfs
Crash	129	341	703
Read-only	64	161	89

Fault Type	Ext3	Ext4
Metadata read failure	70 (66)	95 (90)
Metadata write failure	57 (55)	71 (69)
Metadata corruption	25 (11)	62 (28)
Pointer fault	76 (76)	123 (85)
Interface fault	8 (1)	63 (8)
Memory allocation	56 (56)	69 (68)
Synchronization fault	17 (14)	32 (27)
Logic fault	6 (0)	17 (0)
Unexpected states	42 (40)	127 (54)

stems. *This table shows failures in Ext3, Ext4, and x (3.0 to 3.13).*

Table 2: **Failure Causes in File Systems.** *This table shows the number of different failure causes for Ext3 and Ext4 in Linux 3.5, including those caused by entangled data structures (in parentheses).*

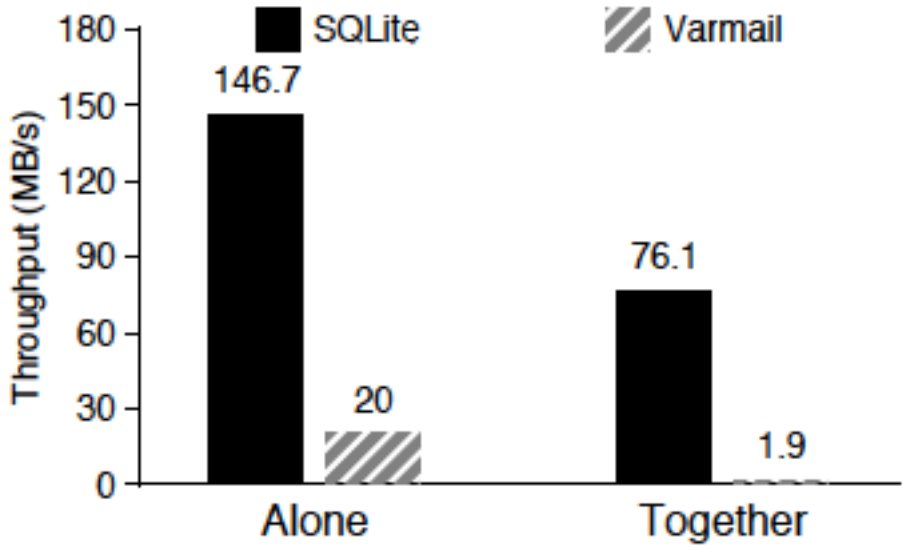
- Entanglement Problems
  - Slow recovery



Scalability of e2fsck on Ext3

Motivation

- Entanglement Problems
  - Bundled performance



SQLite sequentially writes large key/value pairs asynchronously; Varmail frequently calls fsync() after small random writes.

Motivation

- Limitations of current solutions
  - Namespace in FS
    - Provides security in a shared environment to constrain different users or applications
    - Cannot address metadata sharing, transaction entanglement
  - Static disk partition
    - Can isolate corrupted data or metadata
    - Single panic(), BUG_ON() will crash the whole OS
    - Low disk utilization and performance
    - Different to manage a number of file systems

Motivation

- Impact on the higher-level services
  - Virtual Machines
  - Distributed File Systems
    - E.g., HDFS

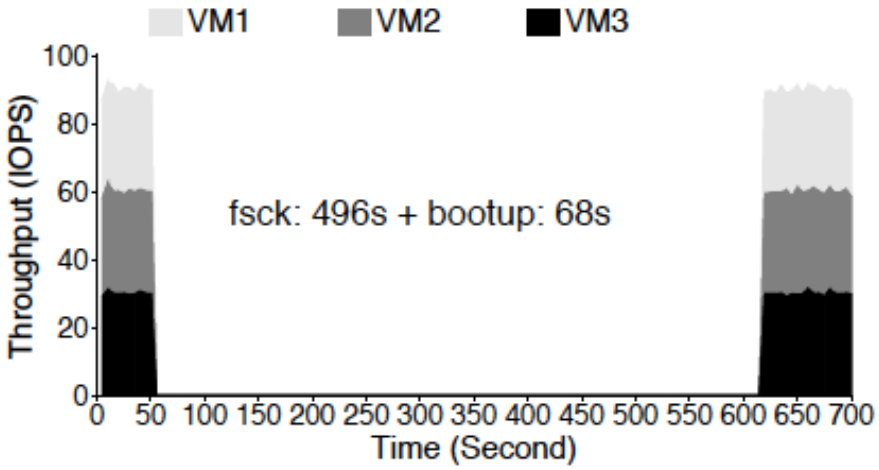


Figure 3: **Global Failure for Virtual Machines.** *This figure shows how a fault in Ext3 affects all three virtual machines (VMs). Each VM runs a workload that writes 4KB blocks randomly to a 1GB file and calls* `fsync()` *after every 10 writes. We inject a fault at 50s, run e2fsck after the failure, and reboot all three VMs.*

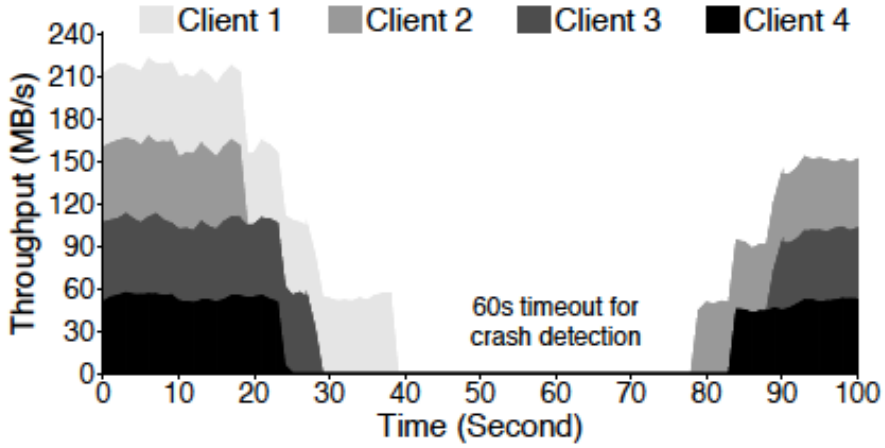


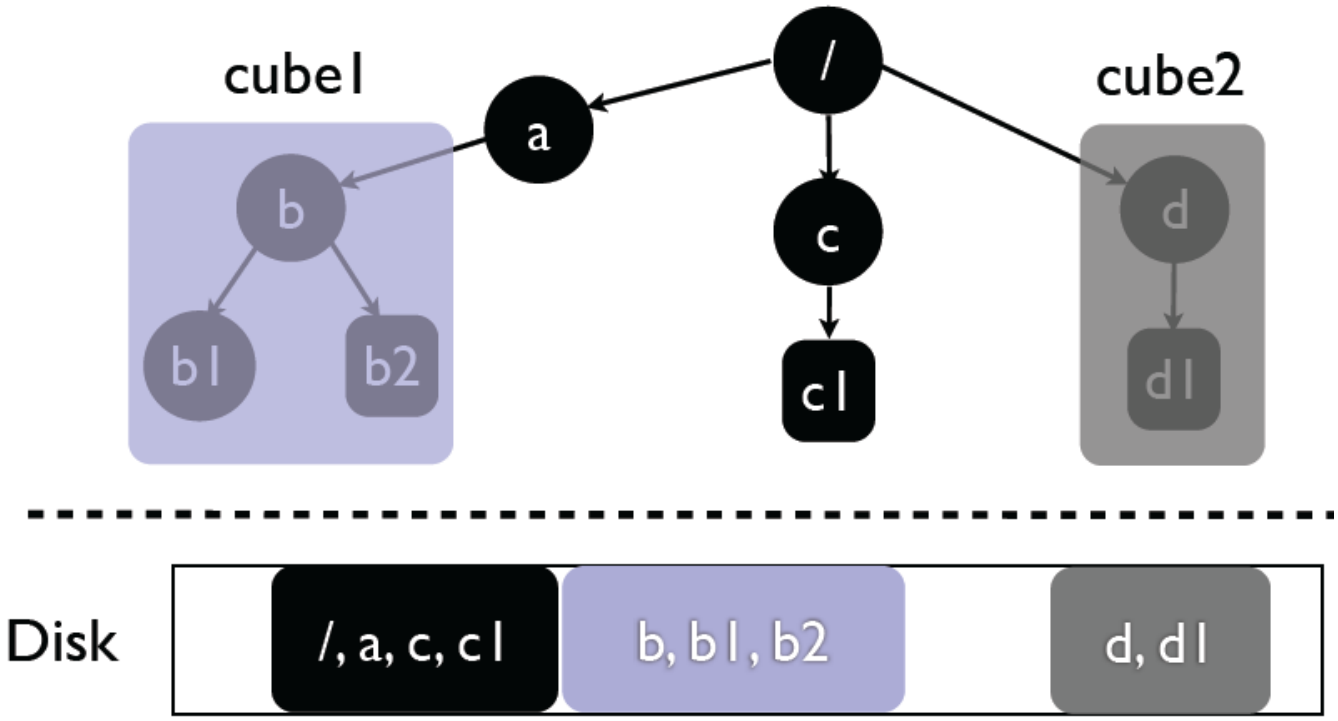
Figure 4: **Impact of Machine Crashes in HDFS.** *This figure shows the negative impact of physical entanglement within local file systems on HDFS. A kernel panic caused by a local file system leads to a machine failure, which negatively affects the throughput of multiple clients.*

File System Disentanglement

- Principles of disentangled data structure
  - No shared physical resources
    - No shared metadata
    - No shared blocks or buffers
  - No access dependency
    - Partition linked lists or trees
    - Avoid directory hierarchy dependency
  - No bundled transactions
    - Use separate transactions
    - Enable customized journaling mode

File System Disentanglement

- A new FS abstraction: Cube
  - An isolated directory in FS, including sub-directory and files
  - Physical disentangled on memory and disk

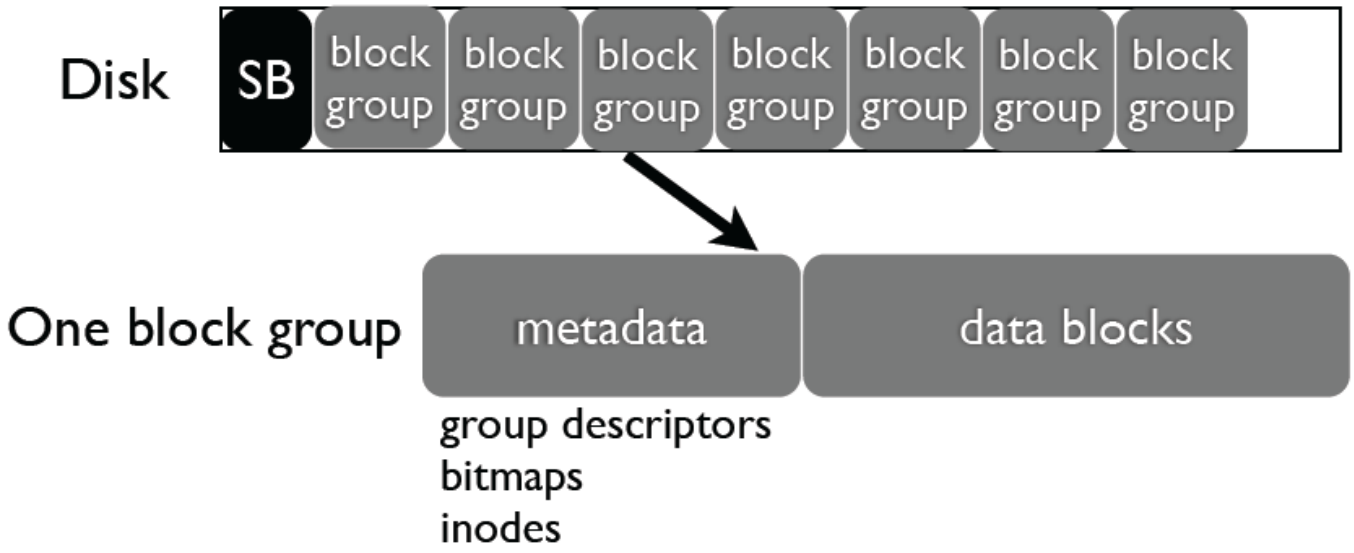


Ice File System (IceFS) Overview

- A disentangled FS that provides the “cube” abstraction
  - Isolated reliability and performance
- Techniques to implement disentanglement
  - Physical resource isolation
  - Access independence
  - Transaction splitting
- A prototype based on Ext3 and JBD
- Benefits:
  - Localized reactions to failures
  - Localized recovery
  - Specialized journaling

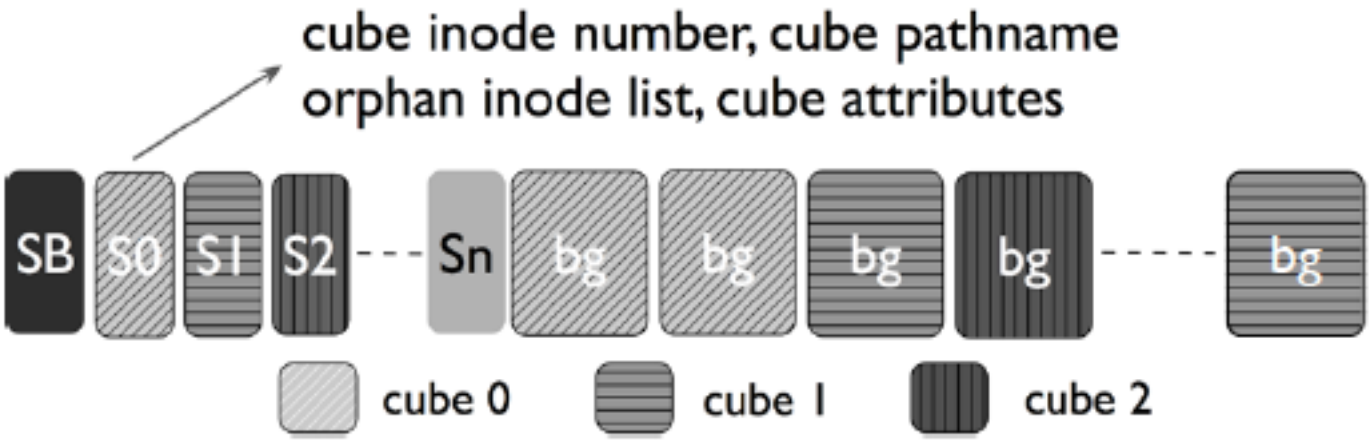
Physical Resource Isolation

- Ext3 Disk Layout
  - A disk is divided into multiple block groups (physically partitioned)



Physical Resource Isolation

- IceFS Disk Layout
  - A block group can be assigned to only one cube at any time.
  - Si is stored in independent block(s).

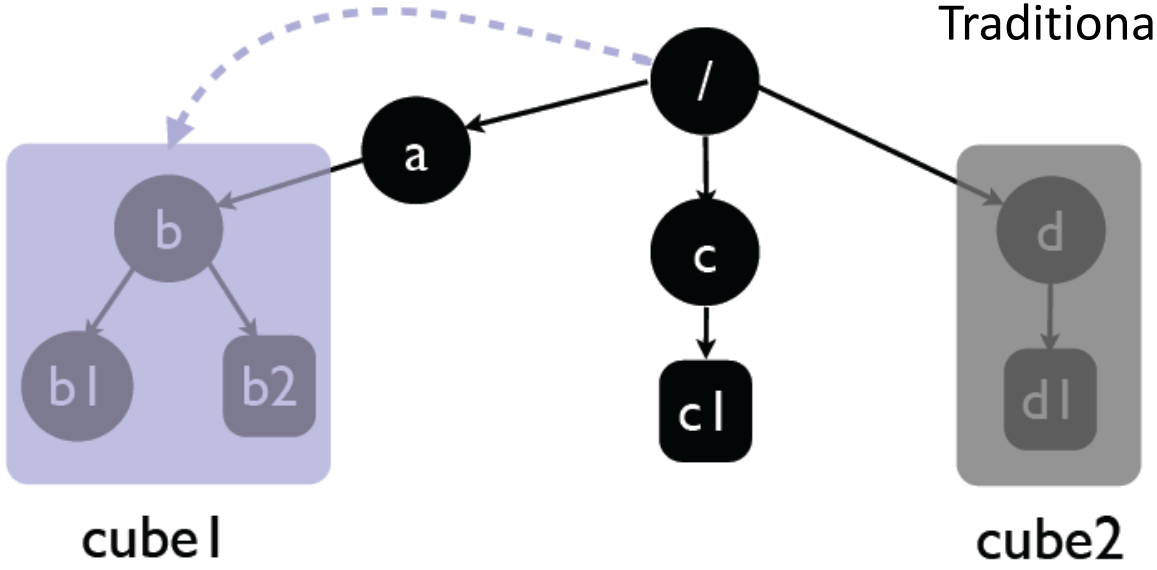


Si: sub-super block for cube i; bg: a block group

Access Independence

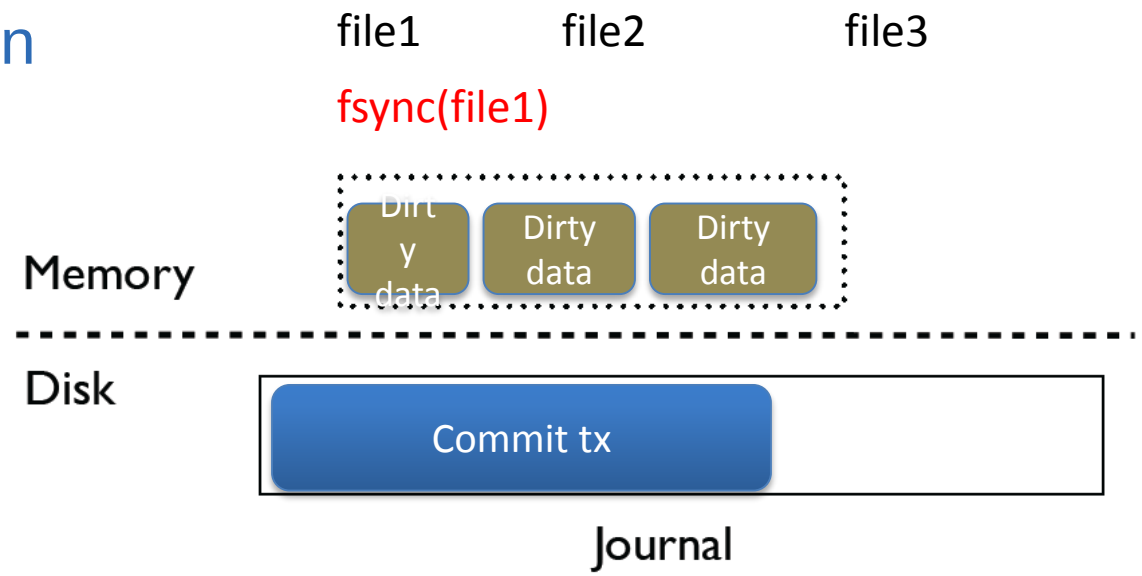
- Principle:
  - No cube can reference another cube.
- Techniques:
  - One sub-super block for each cube stores its own orphan inode list.
  - Directory indirection
- Sub-super block 1,
  Top directory of cube1: /a/b/

Read file /a/b/b1
Prefix match cube1
Jump to cube1 top directory
Traditional pathname lookup

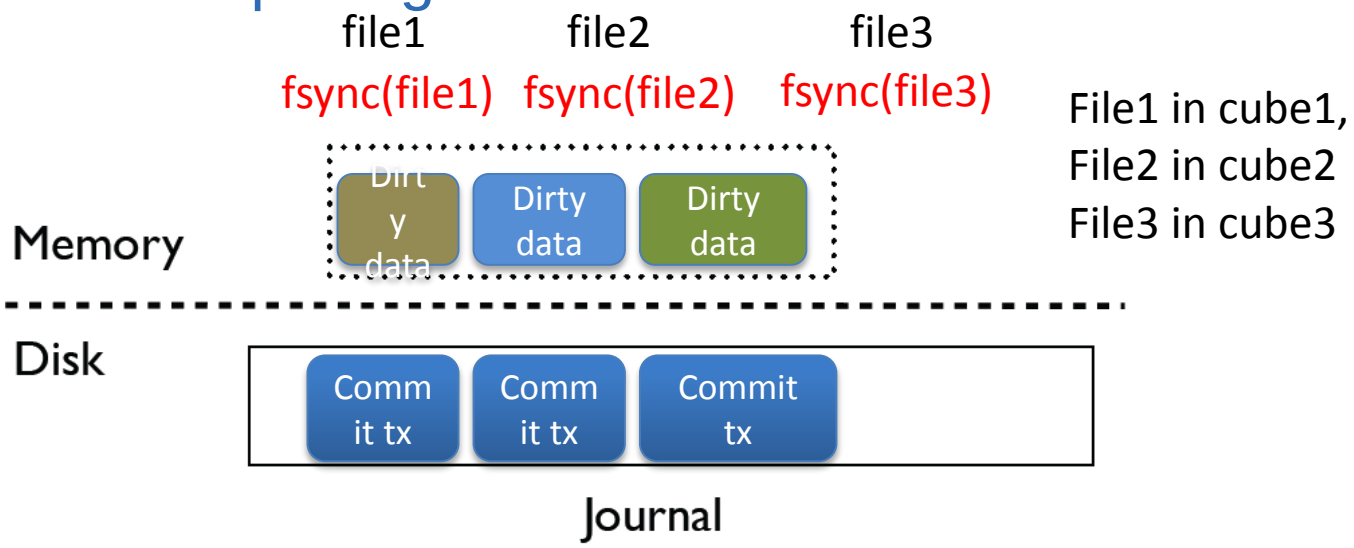


Transaction Splitting

- Ext3 transaction



- IceFS transaction splitting



Benefits of disentangled IceFS

- Localized Reactions to Failures
  - Per-cube read-only and crash
- Localized Recovery
  - Only check faulty cubes
  - Allowing offline and online checking
- Specialized Journaling
  - Concurrent and independent transactions
  - Supporting different journal modes:
    - no fsync, no journal, writeback journal, ordered journal, and data journal

Evaluation

- Overall performance overhead?
  - no failures
- Localize failures?
  - Injure 200 faults
  - Per-cube failure
- Fast Recovery?
  - Cube-aware recovery
- Better journaling performance?
  - Isolated journaling performance
  - Flexibility between consistency and performance
- Useful for scenarios?
  - Virtual machine
  - HDFS

Overall Performance

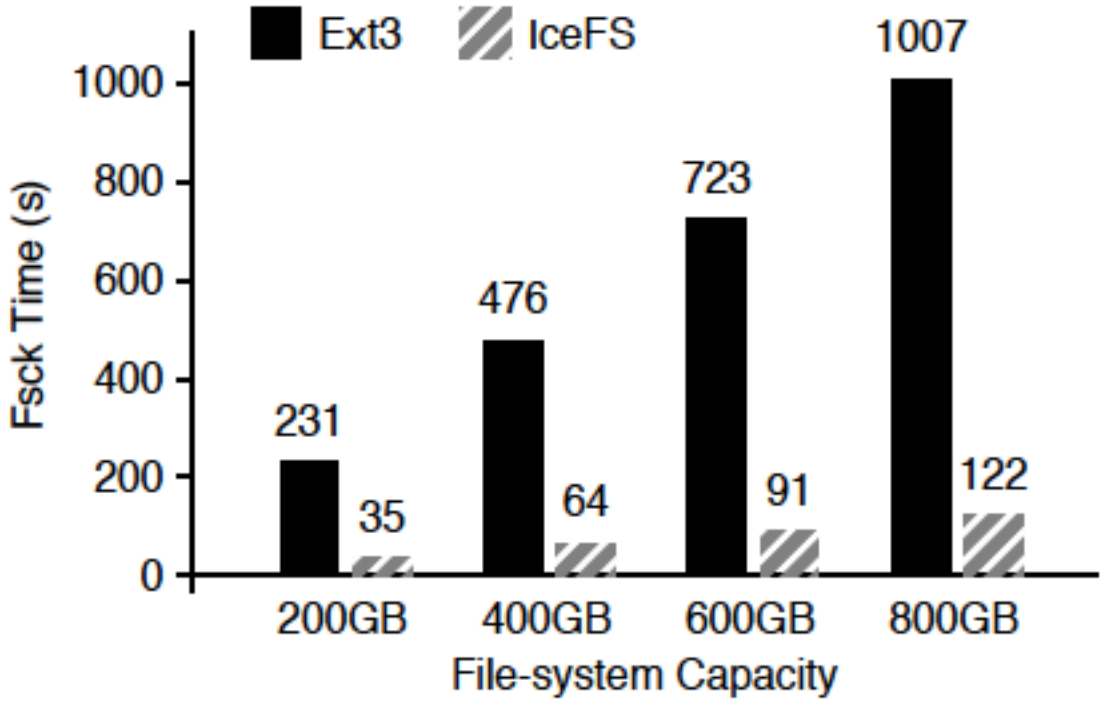
- Micro and Macro benchmarks performance

Workload	Ext3 (MB/s)	IceFS (MB/s)	Difference
Sequential write	98.9	98.8	0%
Sequential read	107.5	107.8	+0.3%
Random write	2.1	2.1	0%
Random read	0.7	0.7	0%
Fileserver	73.9	69.8	-5.5%
Varmail	2.2	2.3	+4.5%
Webserver	151.0	150.4	-0.4%

➢ IceFS incurs little performance overhead.

Fast Recovery

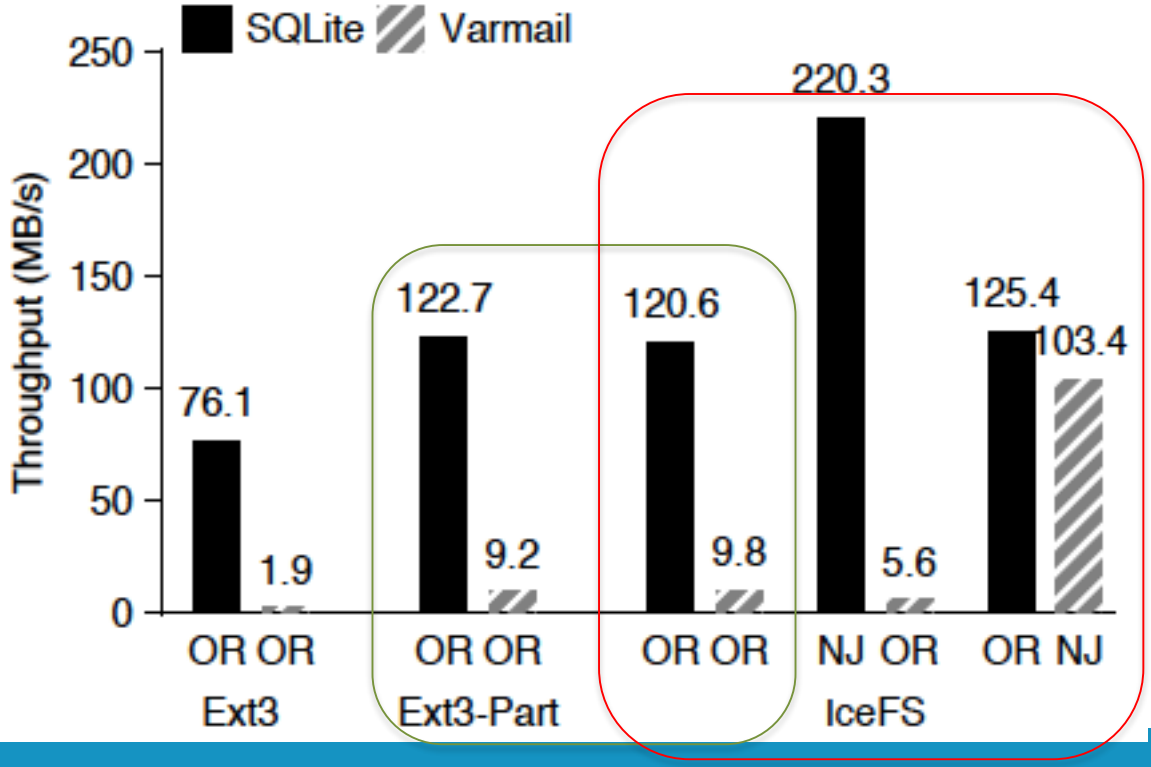
- Performance of IceFS offline fsck



Journal Performance and Flexibility

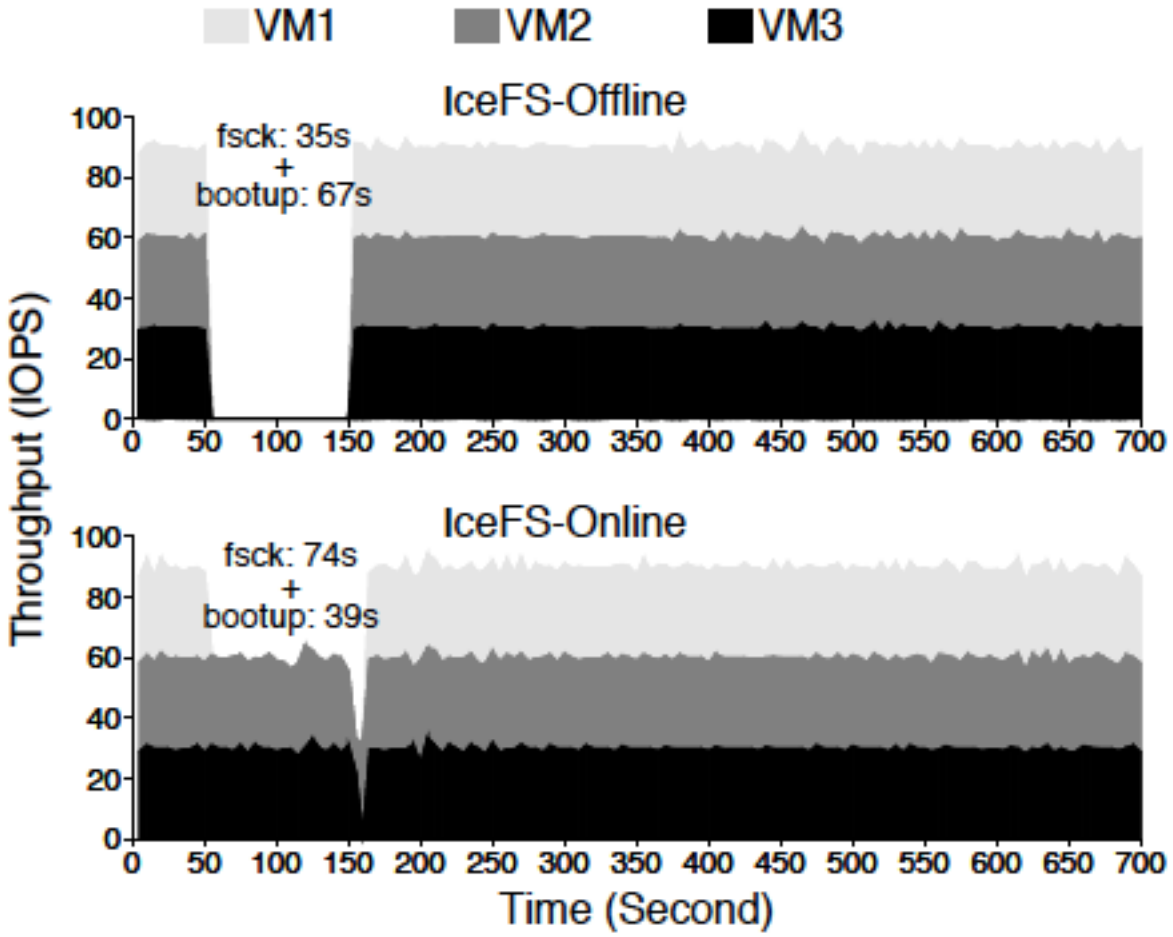
- Running Two Applications on IceFS with Different Journaling Mode

OR: Ordered Journal; NJ: No Journal
Ext3-Part: two separated Ext3 on partitions
SQLite sequentially writes large key/value pairs asynchronously;
Varmail frequently calls fsync() after small random writes.



Usage Scenarios

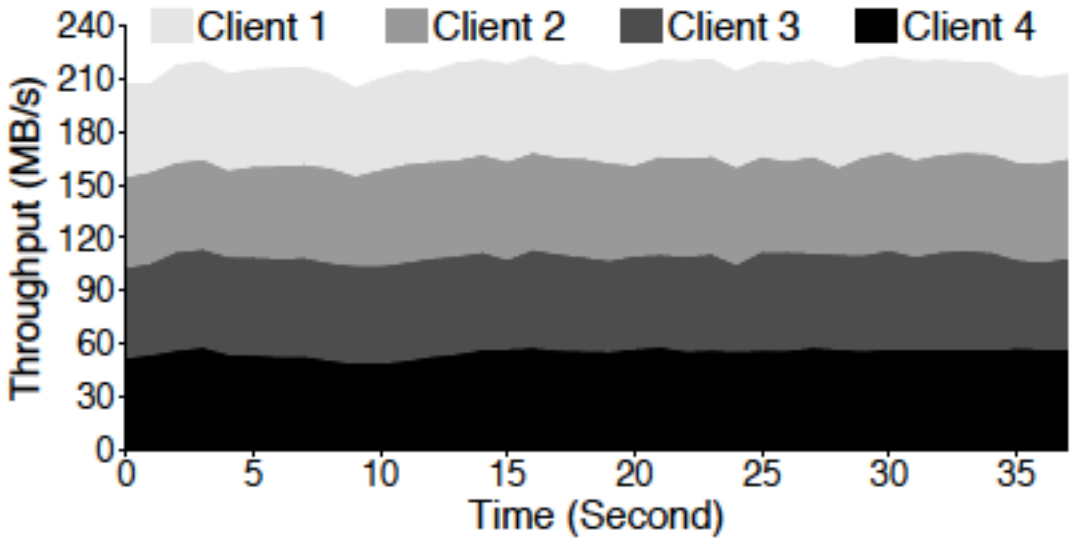
- Virtual Machine



➢ IceFS greatly improves the availability of the VMs compared to that using Ext3.

Usage Scenarios

- HDFS
  - The throughput of 4 different clients when a cube (cube2) failure happens at time 10 second



➢ Impact of the failure to the clients' throughput is negligible.

Related Work

- Fast check
  - Solaris UFS: only check the working-set portion of the FS
  - BSD: background fsck
  - WAFL: Online checking on a volume
  - Chunkfs: partial check of ext2 by partitioning the FS into chunks
  - Changing the I/O pattern of FS checker to avoid random accesses
- Tolerating system crashes
  - Microrebooting: store data state of application components in persistent device outside of the application
  - Nooks: separate address space for each device driver
  - Membrane: tracking resource usage and the requests at runtime
- I/O stack isolation
  - MultiLane: partitioned VFS for scalability

Conclusion

- IceFS: a FS that achieves physical disentanglement (isolation) of data structures.
  - Localized reactions to failures
  - Localized recovery
  - Specialized journaling