Failure-Atomic Updates of Application Data in a Linux File System

--

FAST'2015 short paper

Rajat Verma1 Anton Ajay Mendez1 Stan Park2
Sandya Mannarswamy1 Terence Kelly2
Charles B. Morrey III2

1 HP Storage Division **2** HP Laboratories

夏飞
2015.03.12

Outline

- Introduction
- Failure-Atomic Updates
- Evaluation
- Related Work
- Conclusion

Introduction

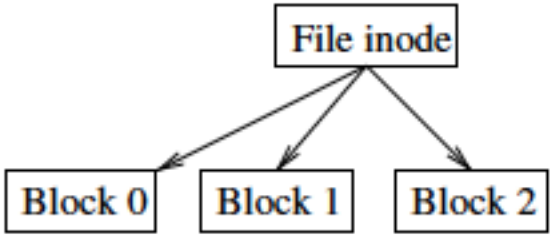
- Consistent modification of application durable data
  - DataBases and Key-Value stores
    - Transaction to guarantee ACID
    - Difficulties: data structure translation, complexity (implementation bugs)
  - File Systems
    - Usually guarantee metadata consistency
    - Data consistency (e.g., data journal mode in ext4):
      - Limitations: not interfaces for applications to specify units of atomic I/O [1]
  - Applications
    - File rename [2]

[1]. Failure-Atomic msync(). EuroSys'2013
[2]. A file is not a file: Understanding the I/O behavior of Apple desktop applications. SOSP'2011

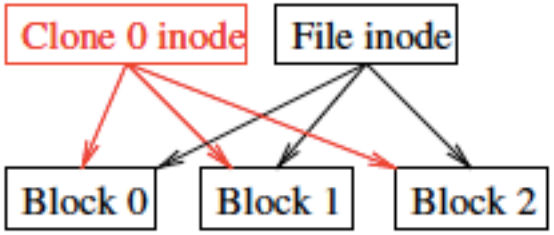
Overview

- Goal
  - Provide failure-atomic updates of application data
- Method
  - Single file atomic updates: O_ATOMIC flag
  - Multi-files atomic updates: *syncv* interface
- Result
  - Correctness of O_ATOMIC
  - Performance: low overhead

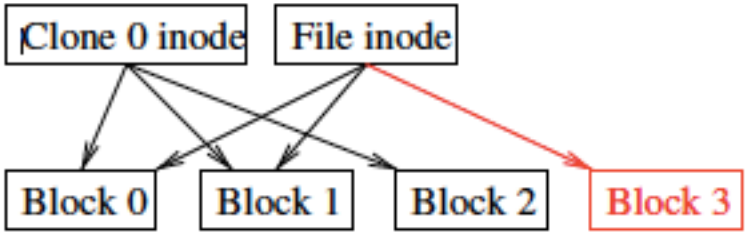
O_ATOMIC



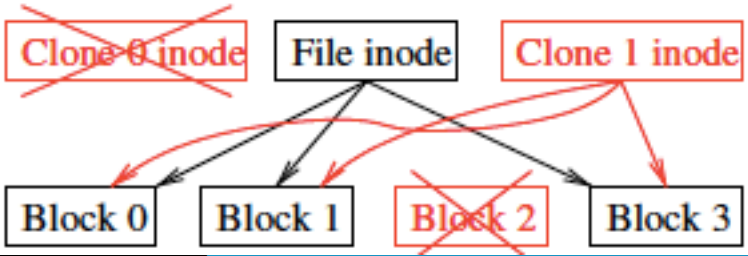
(a) Initial state of file with 3 blocks



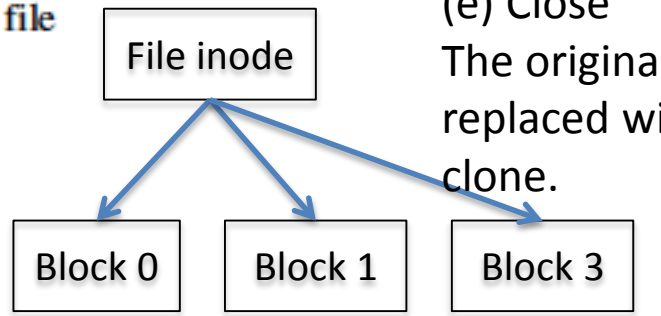
(b) open(O ATOMIC) creates clone



(c) Modifications remap blocks



(d) fsync/msync replaces old clone with new clone

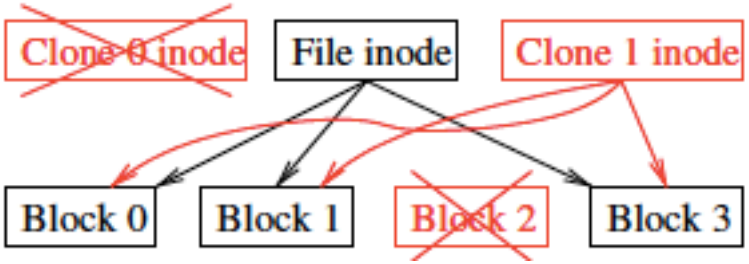


(e) Close
The original file is replaced with the clone.

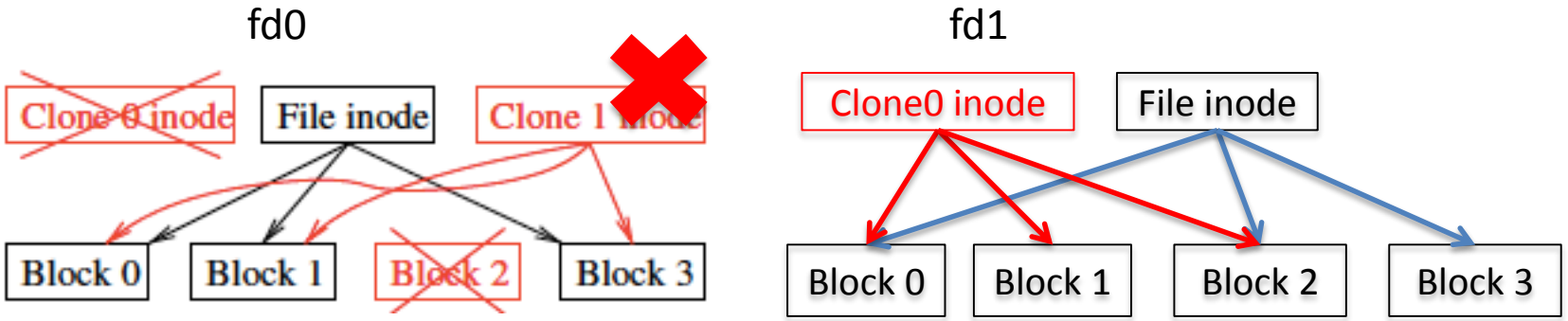
- Crash recovery
  - Check if the clone is existed when the file is accessed again
  - If exist, rename it

Multi-File Atomic Updates: syncv

- Single file fsync/msync



- syncv(fd0, fd1, …)
  - Need to guarantee the atomicity of deleting all the files' clones
  - Method: journaling
    - Metadata modifications required to delete the clones are logged to the journal.



Evaluation

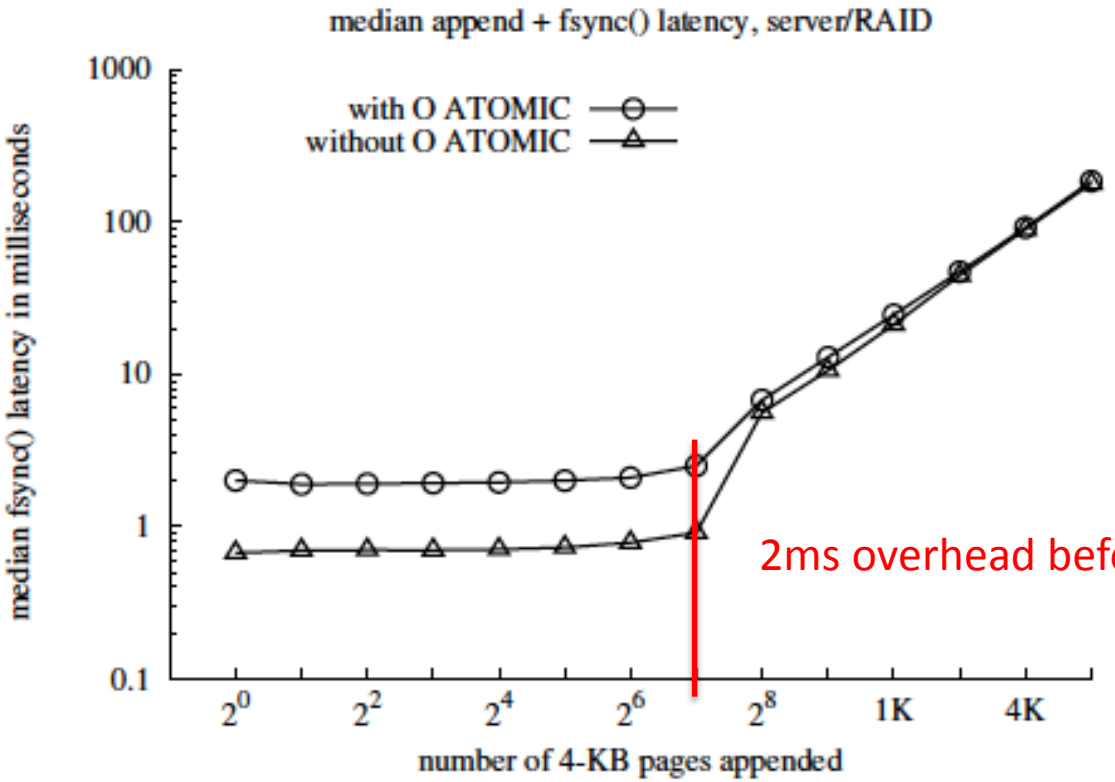
- Correctness of O_ATOMIC
  - Method:
    - Insert crash point into the AdvFS source code.
    - Cut power of a machine
  - Result:
    - Recovery successfully over 400 power interruptions and dozens of crash-points.

Performance

- Platform:
  - Workstation:
    - 2 quadcore 2.4 GHz Xeon E5620 processors, 12 GB of 1333 MHz DRAM,Linux kernel 2.6.32
    - 120GB SATA SSD
  - Server:
    - 12 1.8 GHz Xeon E5-2450L cores and 92 GB of DRAM;
    - 1 GB battery-backed cache configured as 90% write cache
    - 1 TB 7200 RPM SAS hard drive.

Performance

- O_ATOMIC
  - *Write* data to a file followed by *fsync*

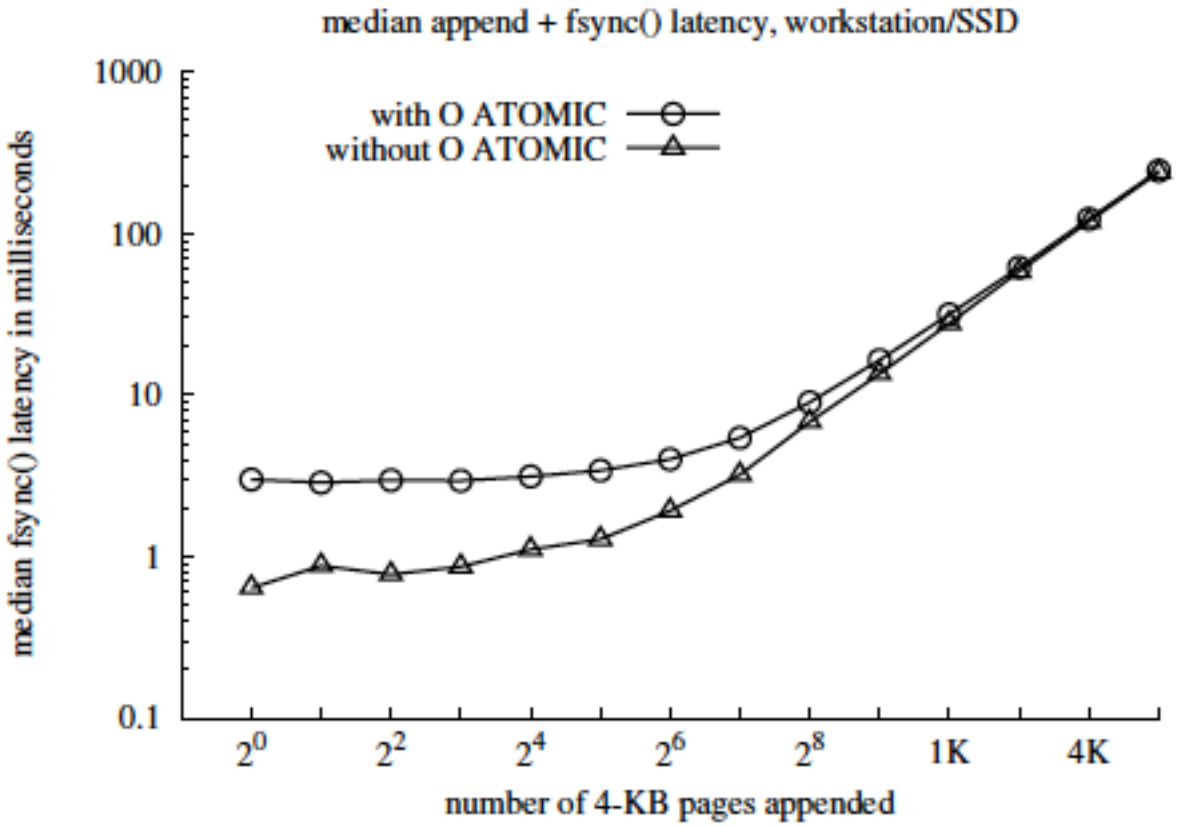


2ms overhead before writing 2^7 pages

Reason: Reading inode from storage to clone with O_ATOMIC.

Performance

- O_ATOMIC



Performance

- Mesobenchmarks: 3,000 transactions
  - insert all keys paired with random1 KB values;
  - replace the value associated with each key with a different random value;
  - finally, delete all of the keys

	Server/RAID			Workstation/SSD		
	insert	replace	delete	insert	replace	delete
STL <map>/AdvFS	1.996	2.488	2.919	1.655	2.022	2.395
Kyoto Cabinet 1.2.76	4.711	2.990	4.660	4.088	2.590	4.007
SQLite 3.7.14.1	2.394	2.524	2.433	2.374	2.611	2.435
LevelDB 1.6.0	0.629	0.626	0.615	0.641	0.640	0.633

Table 1: Mesobenchmarks: Mean per-operation timings (milliseconds).

LevelDB > STL <map>/AdvFS > SQLite > Kyoto Cabinet

Related Work

- Failure-atomic *msync*
  - Only apply to memory-mapped file
  - Data modifications are written twice due to journaling
- Fusion-io atomic-write
  - Special hardware, only apply to single-file updates, cannot address updates to memory-mapped file
- Vista Transactional FS (TxF)
  - Deprecated due to complex interface
- Transaction OS (TxOS)
  - Implemented by FS journal: write twice, transaction size
- Works on persistent memory
  - Mnemosyne: do not support conventional FS operations
  - Software persistent memory (SoftPM): 512KB granularity
- CoW FS
  - Conventional: ZFS (bubbling up to the root)
  - Optimized: BPFS (short-circuit shadowing page)

Conclusion

- Provide interfaces for applications to guarantee failure-atomic updates.
  - O_ATOMIC flag
  - *syncv()*
- Simple and efficient