Towards Efficient, Portable Application-Level Consistency

Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram,

Joo-Young Hwang, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

WISDOM

File System
Crash Consistency

File System Crash Consistency

What happens if there is a system crash during a file system update?

File system crash consistency: Make sure file system's metadata is logically consistent, even if there is a crash

Multiple techniques: FSCK, Soft Updates, Journaling, Copy-On-Write …

Application-Level
Crash Consistency (ALC)

Application-Level Consistency (ALC)

What happens to user data if there is a crash?

Consistency of user data – Application-Level Crash Consistency (ALC)

This work – Study of what happens to user data

Result

Result

State of the art: For effective application-level consistency, application developers depend on specific details of file system implementation

This is bad: Many file systems in use (Linux: ext3, ext4, btrfs, xfs, zfs …)

Outline

Background: Application-Level Consistency (ALC)

Goals, Methodology of Study

File System Behavior

ALC Bugs

ALC Performance

Summary

Outline

Background: Application-Level Consistency (ALC)

Goals, Methodology of Study

File System Behavior

ALC Bugs

ALC Performance

Summary

Application-Level Data Structures

Modern applications store many data structures

Google Chrome initialization: 500+ files

- History
- Cookies
- Web page cache

Application-Level Consistency (ALC)

- Applications impose invariants on data
  - Web page cache: Should contain complete entries
  - Photo application: Thumbnails match pictures
- Invariants should hold across system crashes
  - Violation: application failures, silent corruption
- Requires complex implementations
  - eXplode [OSDI '06], Eat My Data [Stewart Smith]

Example: Atomic File Rewrite

File *grub.conf* (Original)

```
kernel vmlinuz
initrd initrd.img
```

→

File *grub.conf* (Updated)

```
print "Hello"
kernel vmlinuz
initrd initrd.img
```

File should always be in either

- Fully original state or fully updated state

File should never

- Contain garbage, or be empty, or filled with zeroes

Atomic File Rewrite – Correct Protocol

```
fd = creat("temp")
write(fd)
fsync(fd)
rename("temp","grub.conf")
```

Atomic File Rewrite – Wrong Protocol

fd = creat("temp")
write(fd)
~~fsync(fd)~~
rename("temp","grub.conf")

Occurs because file systems can re-order write() and rename()

Possible states after crash

grub.conf (Original)	*grub.conf* (Updated)	*grub.conf* (Zeroes)
	print "Hello"	000000000000000
kernel vmlinuz	kernel vmlinuz	000000000000000
initrd initrd.img	initrd initrd.img	000000000000000

Atomic File Rewrite – Depends on FS

Wrong protocol is commonly used – why?

- Bug (invalid assumption)
- Correctness sacrificied for performance

Works under most common file systems

- Ext4, btrfs etc. explicitly ensure correctness
- Though not required by standard FS interface

Observation:

- FS implementation affects applications

Outline

Background: Application-Level Consistency (ALC)

Goals, Methodology of Study

File System Behavior

ALC Bugs

ALC Performance

Summary

Goals

- Study relationship of FS implementation with
  - ALC correctness
  - ALC performance
- Characterize common file systems
  - Deduce high-level "properties" affecting ALC

Methodology

- Case study: Two applications (SQLite, LevelDB)
  - Find new bugs, analyze existing bugs
    - Manual system call trace analysis, Bugzilla
  - Find any correctness-performance tradeoffs
- Extract FS implementation details affecting bugs
- Convert details to high-level properties
- Characterize file systems
  - Understanding source code

Outline

Background: Application-Level Consistency (ALC)

Goals, Methodology of Study

File System Behavior

ALC Bugs

ALC Performance

Summary

Post-Crash Property

Post-Crash Property (True / False):

Does a system call sequence only result in a given, desirable set of post-crash states

1. fd = creat("FileA.temp")
2. write(fd)
3. ~~fsync(fd)~~
4. rename("FileA.temp", "FileA")

Safe-rename property

grub.conf (Updated)

print "Hello"
kernel vmlinuz
initrd initrd.img

(or)

grub.conf (Original)

kernel vmlinuz
initrd initrd.img

~~*grub.conf* (Garbage)~~

~~#!@$%#!@$%#!~~
~~@$%#!@$%#!@~~
~~$%#!@$%#!@$%~~

~~(or)~~

~~*grub.conf* (Zeroes)~~

~~000000000000000~~
~~000000000000000~~
~~000000000000000~~

File System Comparison

Different configurations of ext3 file system

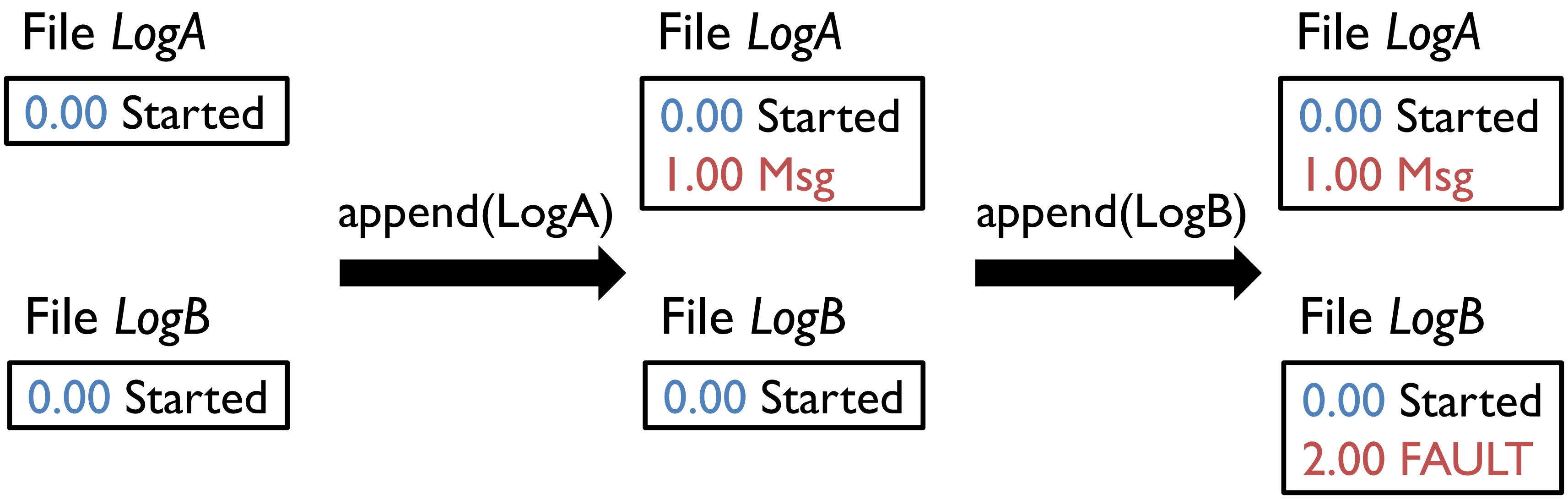
Different versions of ext4 file system

	Safe rename
ext3 – ordered	✓
ext3 – writeback	✓
ext4 – ordered	✓
ext4 – ordered – original version	
btrfs	✓

Ordered Appends

Ordered appends property

1. Append(LogA)
2. Append(LogB)



Ordered Appends

Ordered appends property

1. Append(LogA)
2. Append(LogB)

File *LogA*

0.00 Started

File *LogB*

0.00 Started

(or)

File *LogA*

0.00 Started
1.00 Msg

File *LogB*

0.00 Started

(or)

File *LogA*

0.00 Started
1.00 Msg

File *LogB*

0.00 Started
2.00 FAULT

File *LogA*

0.00 Started

File *LogB*

0.00 Started
2.00 FAULT

File System Comparison

Ordered appends: Appends get persisted in the issued order

	Safe rename	Ordered appends
ext3 – ordered	✓	✓
ext3 – writeback	✓	
ext4 – ordered	✓	
ext4 – original		
btrfs	✓	

More Properties

Ordered dir-ops: Directory operations (creat, unlink, rename …) get persisted in issued order

Safe appends: When a file is appended, the appended portion will never contain garbage

Safe new file: After fsync() on a new file, another fsync() on the parent directory is not needed

File System Comparison

Ext3-ordered: Safest for applications

Safe new file: Manpages explicitly warn against this property.

	Safe rename	Ordered appends	Ordered dir-ops	Safe appends	Safe new file
ext3 – ordered	✓	✓	✓	✓	✓
ext3 writeback	✓		✓		✓
ext4 – ordered	✓		✓	✓	✓
ext4 –original			✓	✓	✓
Btrfs	✓			✓	✓

Outline

Background: Application-Level Consistency (ALC)

Goals, Methodology of Study

File System Behavior

ALC Bugs

ALC Performance

Summary

Bugs: LevelDB Guarantees

LevelDB is a key-value database

Put(key, value, *synchronous*)

- Atomic
- Ordered
  - If a *Put()* can be retrieved, all previous *Put*() can also be retrieved
- *Synchronous* = *true*: Durable
- No corruption

Bugs: Guarantees *vs* Post-Crash Properties

Post-Crash Property	LevelDB Guarantee Affected
Ordered append	Re-ordering, Corruption
Ordered directory operations	Re-ordering, Corruption
Safe new file	Corruption
Safe rename	Corruption (Previously fixed bug)

Outline

- Background: Application-Level Consistency (ALC)
- Goals, Methodology of Study
- File System Behavior
- ALC Bugs
- ALC Performance
- Summary

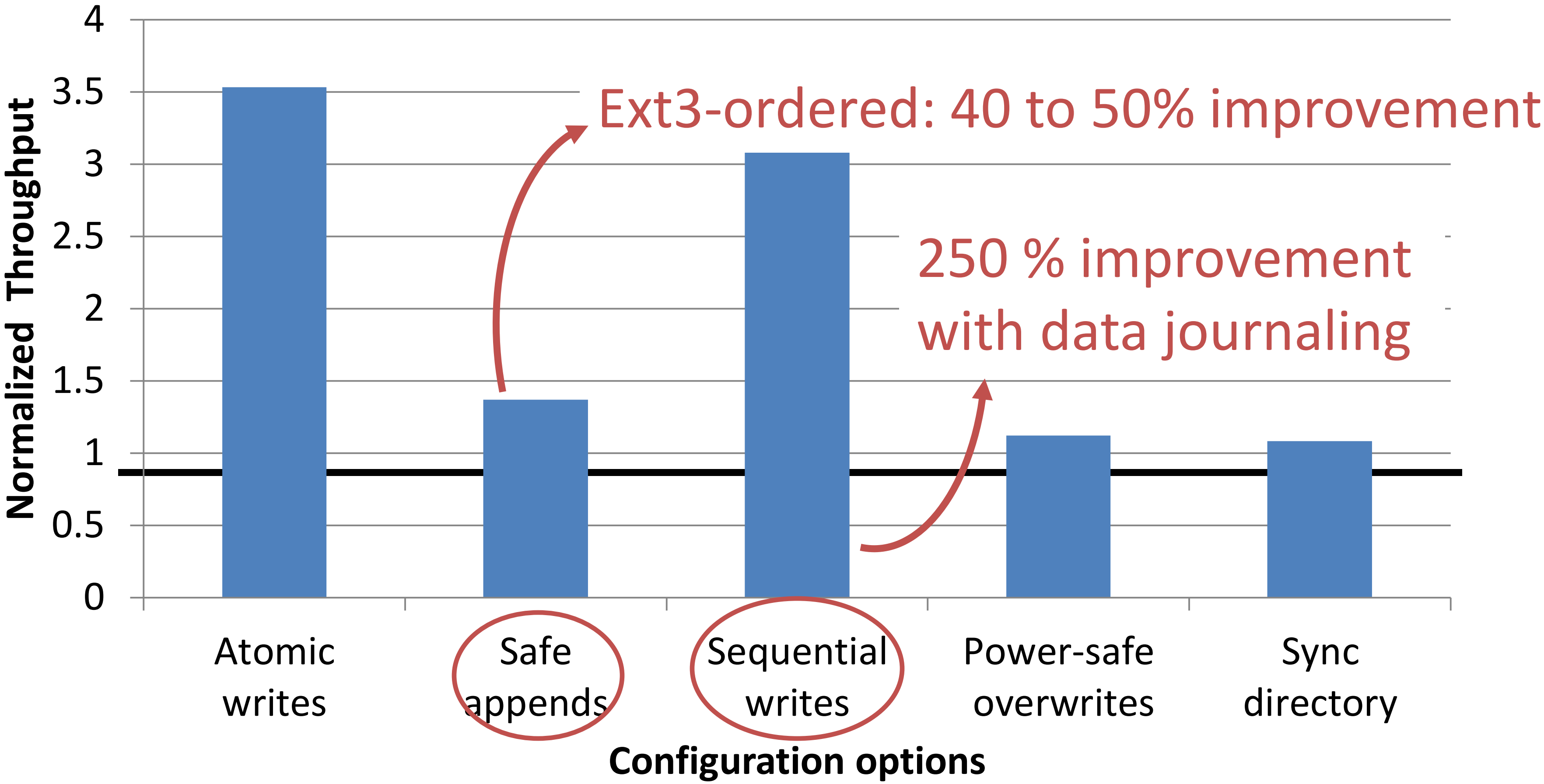
Performance Optimizations

SQLite:

> "In particular, we suspect that most modern filesystems exhibit the safe append property and that many of them might support atomic sector writes. But until this is known for certain, SQLite will take the conservative approach and assume the worst."

- Five configuration options
- Evaluated performance for each option
  - On top of ext3 - ordered

Performance: SQLite – Configurations



Outline

Background: Application-Level Consistency (ALC)

Goals, Methodology of Study

File System Behavior

ALC Bugs

ALC Performance

Summary

Summary

Bugs

- Four new bugs in LevelDB
- Past bugs: One in LevelDB, three in SQLite
- All bugs exposed on some file system

Performance

- Wildly differing performance when optimized for exact file system behavior

Conclusion

State of the art: For effective application-level consistency, application developers depend on specific details of file system implementation

Thank you!

Questions or suggestions?

UNIVERSITY OF WISCONSIN–MADISON

WISDOM