WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

WISDOM

Application-Level Crash Consistency

Thanumalayan Sankaranarayana Pillai, Ramnatthan Alagappan, Vijay Chidambaram, Samer Al-Kiswany, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau

File System
Crash Consistency

File System Crash Consistency

If the system crashes during a file system update…

Ensure file system metadata is logically consistent

Techniques: FSCK, Soft Updates, Journaling, etc.

Application-Level Crash Consistency (ALC)

Application-Level Consistency (ALC)

What happens to user data during a crash?

Consistency of user data: ALC

This work: Study of what happens to user data

- 12 applications
- BerkeleyDB, HDFS, ZooKeeper, VMWare Player

Result

Result

ALC depends on specific details of file system implementation

- 65 vulnerabilities across 12 applications
- All studied applications have vulnerabilities

Bad situation: Many file systems in use

- Linux: ext3, ext4, btrfs, xfs etc.

Outline

Background

Framework

Study

What is ALC?

Consistency of user data during a crash

Example:

SQLite without ALC

(No ACID properties during crash)

```
write(home/file.db)
write(home/file.db)
```

What is ALC?

Consistency of user data during a crash

Example:

SQLite without ALC

(No ACID properties during crash)

```
write(home/file.db)
write(home/file.db)
```

SQLite with ALC

(ACID during system crash and process crash)

```
creat(home/journal)
append(home/journal)
      ...
fsync(home/journal)
fsync(home)
append(home/journal)
fsync(home/journal)
write(home/file.db)
fsync(home/file.db)
unlink(home/journal)
```

<u>Update protocol</u>

ALC is a Complex Problem

Update protocol needs to be highly optimized

- Involves *fsync()*, usually a performance bottleneck

Crash recovery rarely invoked

- Updated protocol mostly untested

<u>ALC deals with hidden disk state</u>

Disk State

Application-level I/O modifies buffer cache

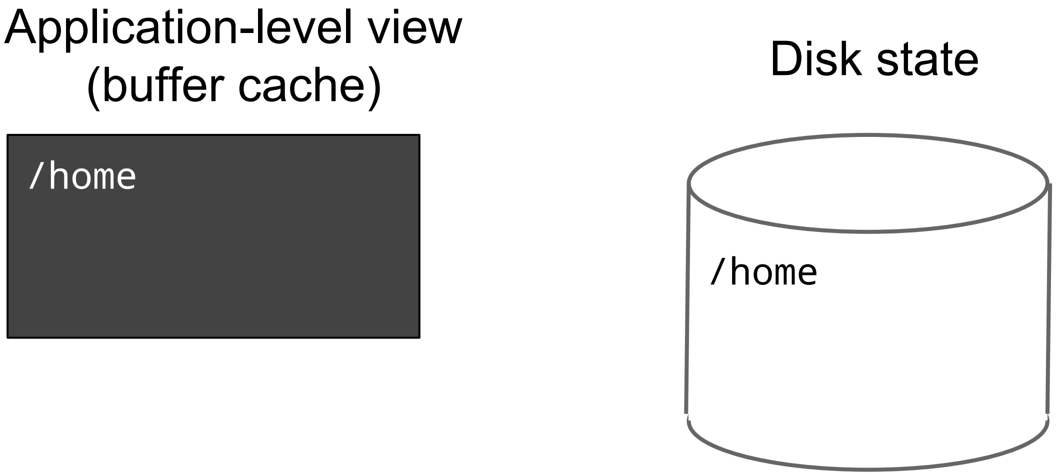
File system slowly persists buffer cache to disk

Disk state: State recovered by file system after a crash

Disk State

Application-level I/O modifies buffer cache

File system slowly persists buffer cache to disk



Application-level view (buffer cache)

/home

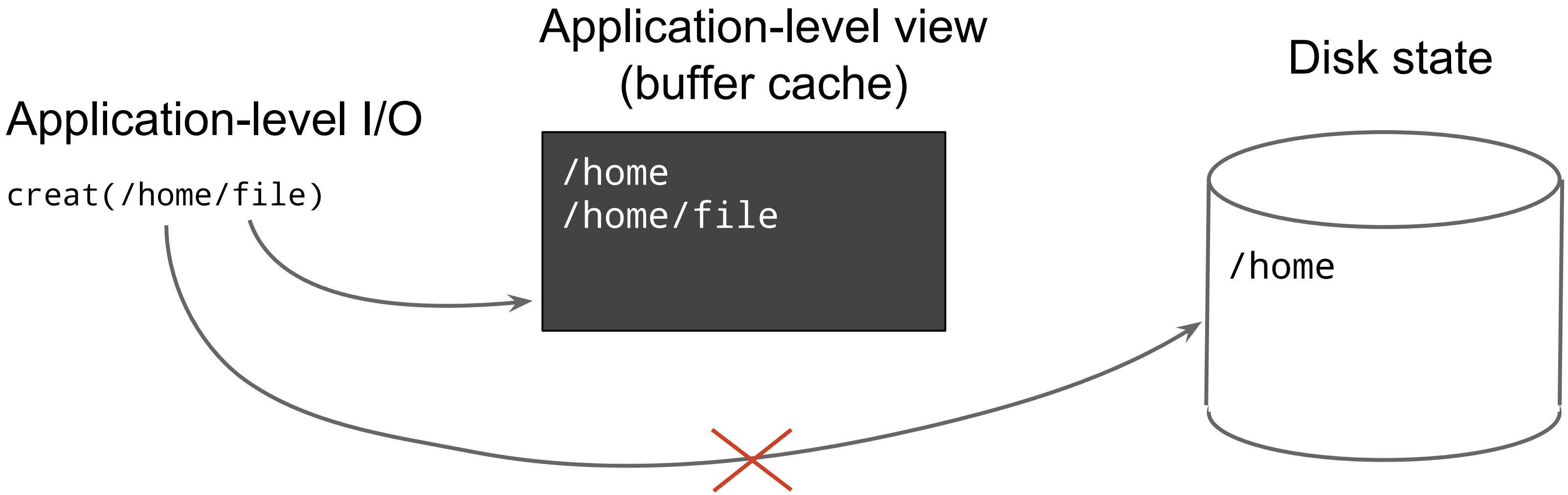
Disk state

/home

Disk State

Application-level I/O modifies buffer cache

File system slowly persists buffer cache to disk



Application-level I/O

```
creat(/home/file)
```

Application-level view
(buffer cache)

```
/home
/home/file
```

Disk state

```
/home
```

Application View vs Disk State: The Difference

Durability

Ordering

Atomicity

Application View vs Disk State: The Difference

Durability

Ordering

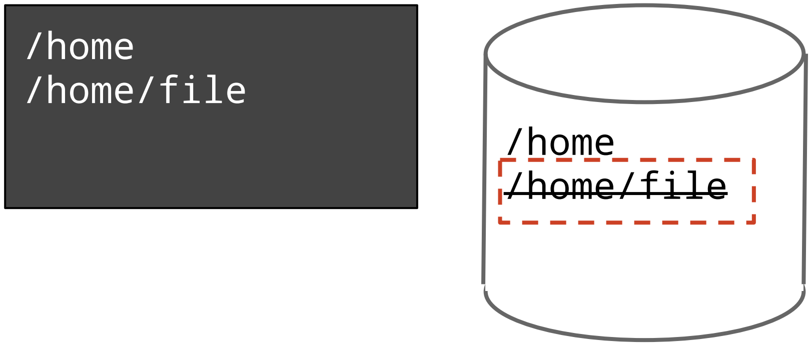
Atomicity

Application I/O

```
creat(/home/file)
```

Application view

Disk state



```
/home
/home/file
```

/home

~~/home/file~~

Application View vs Disk State: The Difference

Durability

Ordering

Atomicity

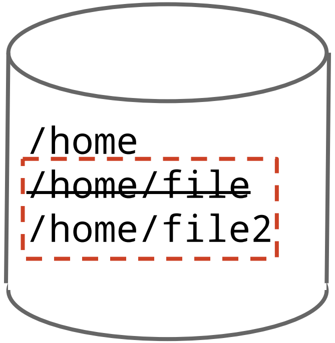
Application I/O

```
creat(/home/file)
creat(/home/file2)
```

Application view

```
/home
/home/file
/home/file2
```

Disk state



/home

~~/home/file~~

/home/file2

Application View vs Disk State: The Difference

Durability

Ordering

Atomicity

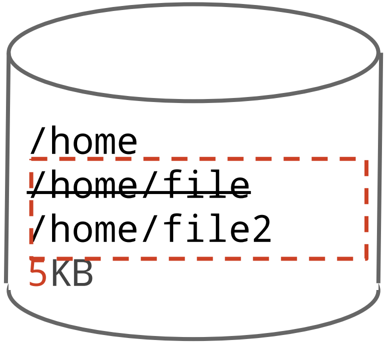
Application I/O

```
creat(/home/file)
creat(/home/file2)
write(/home/file2,10KB)
```

Application view

```
/home
/home/file
/home/file2 10KB
```

Disk state



/home

~~/home/file~~

/home/file2

5KB

Persistence Properties

File systems vary on ordering, atomicity behavior

- ALC correctness depends on file systems

Persistence properties: Ordering and atomicity properties of file systems

- Example: Ordering between directory operations (create, unlink, rename...)

Preliminary unverified results. Do not use for analysis or conclusions.

Persistence Properties

File Systems

File System		Ordering			Write atomicity	
		File writes	File and directory ops	Directory ops	Block	Multi-Block
ext2		X	X	X	X	X
ext3	data=writeback	X	X		X	X
	data=ordered	X			X	X
	data=journal					X
ext4	data=writeback	X	X		X	X
	data=ordered	X	X		X	X
Btrfs		X	X	X		X
XFS		X	X		X	X
ReiserFS		X			X	X

Background: Summary

Implementing ALC is complex

- File systems vary on ordering and atomicity behavior
- Update protocols are untested

Few studies on ALC vulnerabilities

Outline

Background and Motivation

Framework

Study

Framework - Overview

1. Collect application-level trace
2. Calculate possible crash states
3. Check ALC on each crash state

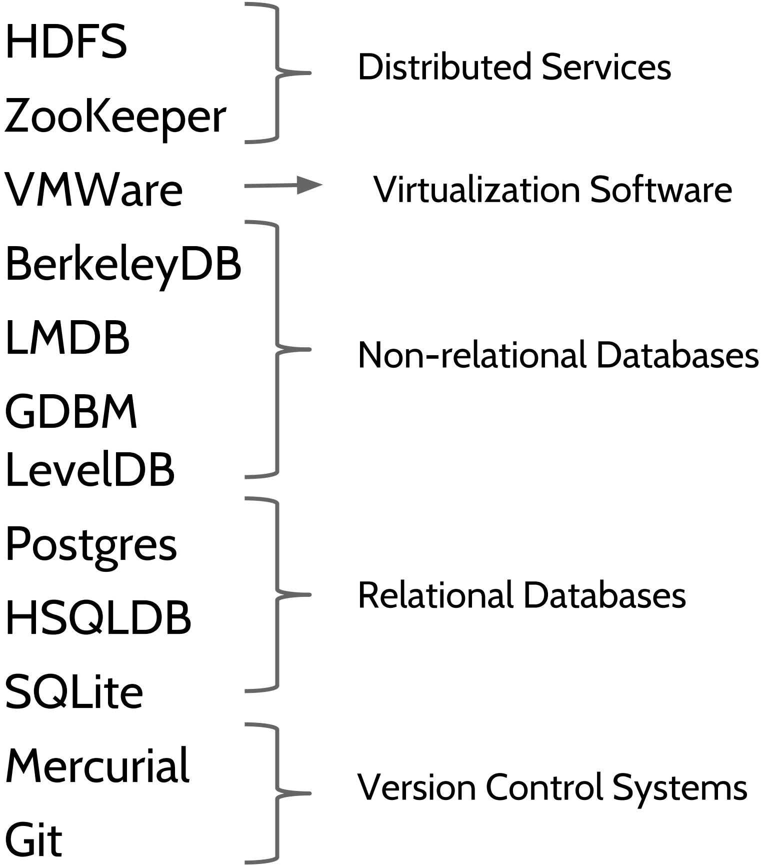
Outline

Background and Motivation

Framework

Study

Applications



HDFS

ZooKeeper

Distributed Services

VMWare

Virtualization Software

BerkeleyDB

LMDB

GDBM

LevelDB

Non-relational Databases

Postgres

HSQLDB

SQLite

Relational Databases

Mercurial

Git

Version Control Systems

Example: ZooKeeper

mkdir(v)

creat(v/log)

…

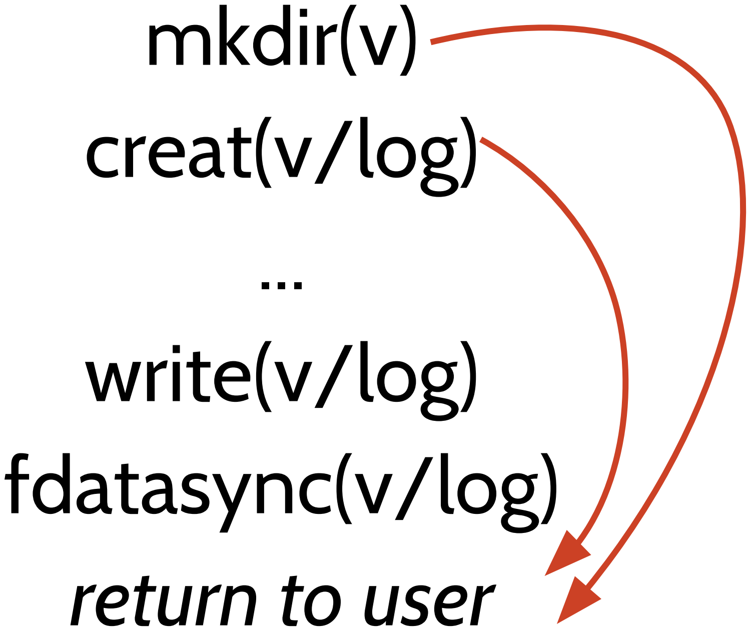
write(v/log)

fdatasync(v/log)

return to user

Example: ZooKeeper

Preliminary unverified results. Do not use for analysis or conclusions.



mkdir(v)

creat(v/log)

…

write(v/log)

fdatasync(v/log)

return to user

Example: ZooKeeper

Preliminary unverified results. Do not use for analysis or conclusions.

mkdir(v)

creat(v/log)

...

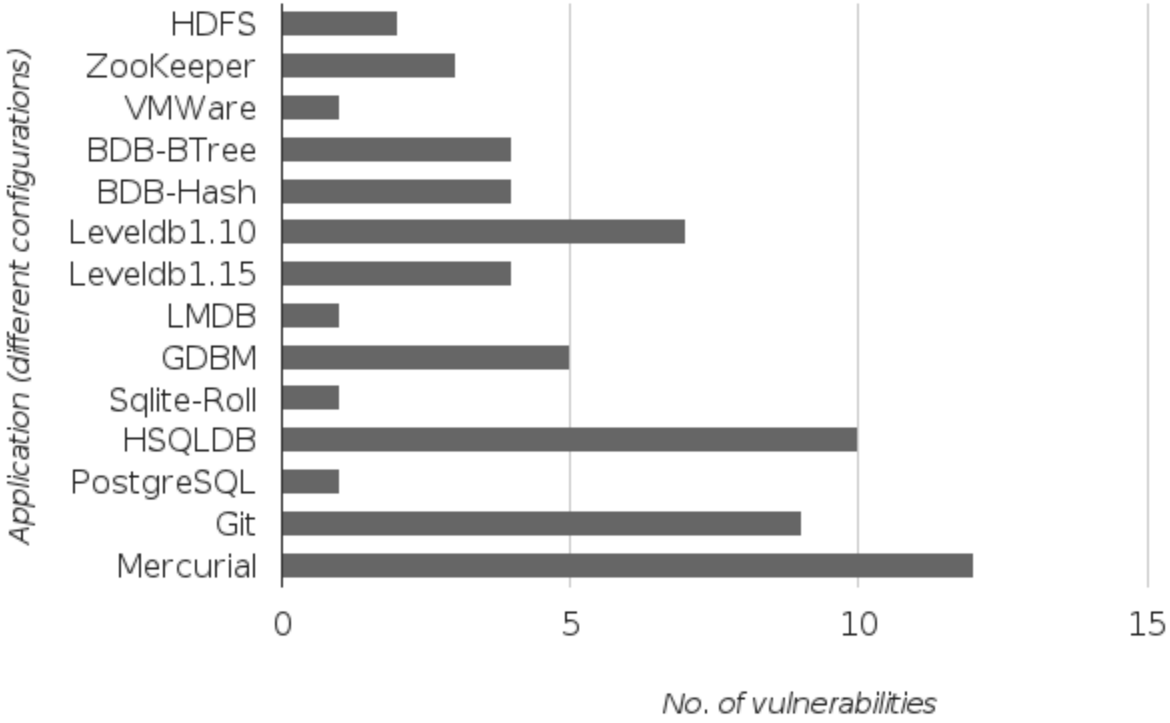
write(v/log)

fdatasync(v/log)

return to user

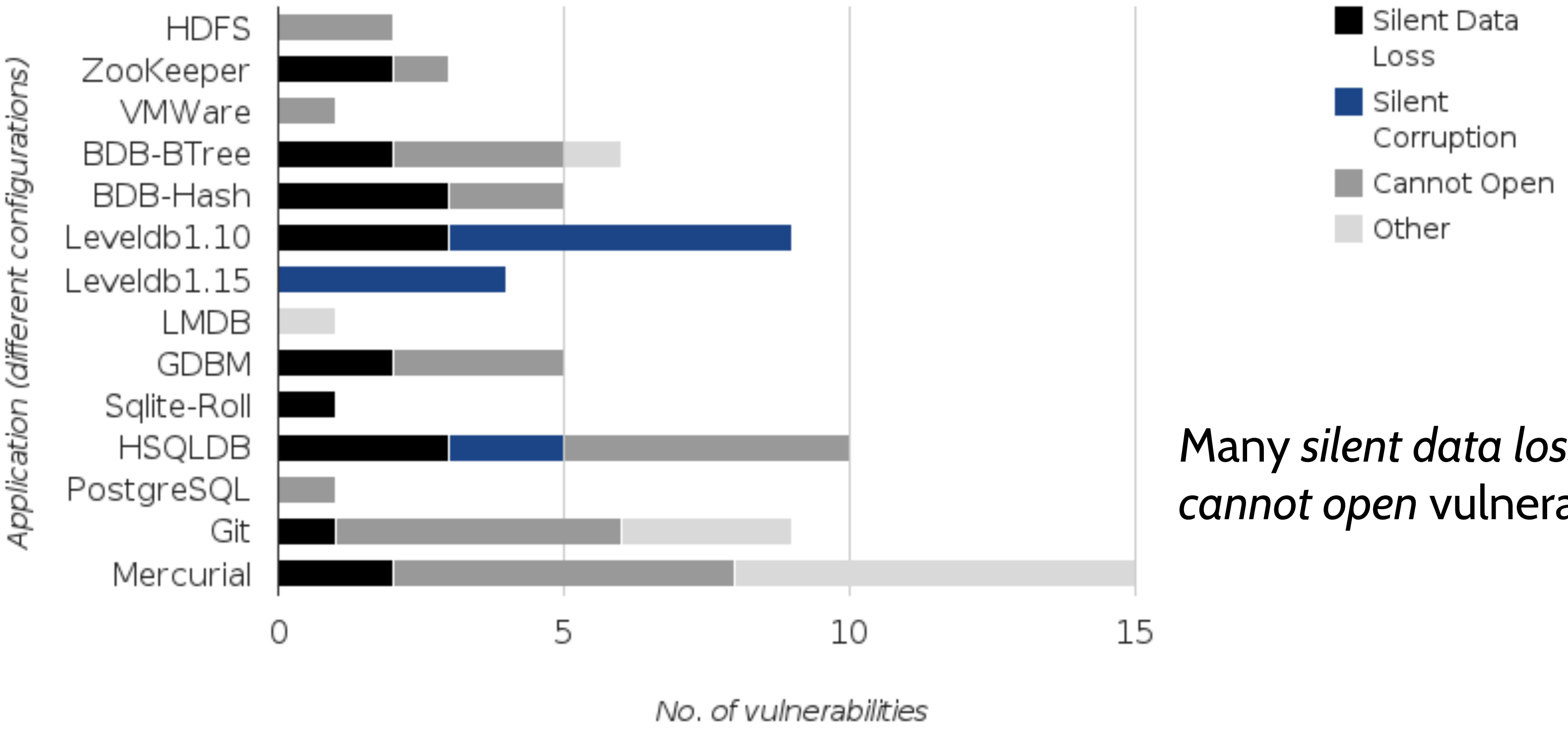
Vulnerabilities

Preliminary unverified results. Do not use for analysis or conclusions.



Vulnerabilities: Consequences

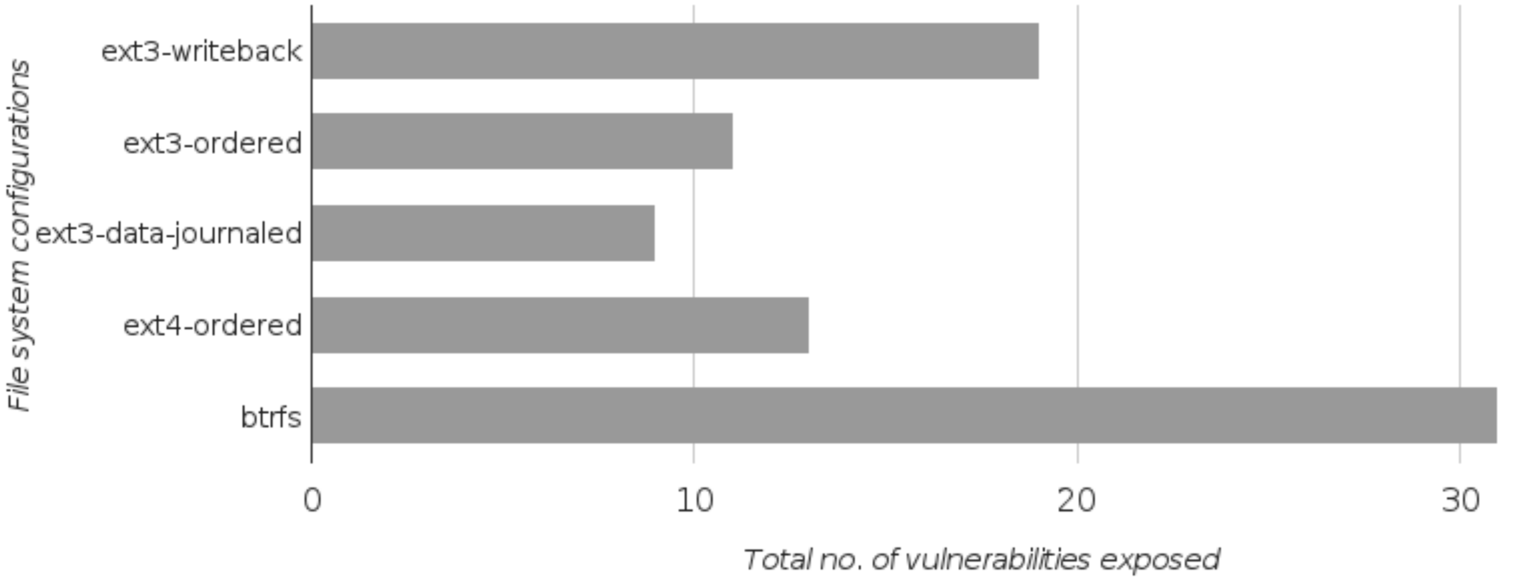
Preliminary unverified results. Do not use for analysis or conclusions.



Many *silent data loss*, *cannot open* vulnerabilities

Vulnerabilities per File System

Preliminary unverified results. Do not use for analysis or conclusions.



Patterns

Preliminary unverified results. Do not use for analysis or conclusions.

Appends need to be atomic

- Because of implementations of write-ahead logging
- Overwrites mostly don't need to be

Append-before-rename only improves correctness lightly

- Might help more with different class of applications

A file system design that is fast, but helps ALC

Summary

ALC is dependent on file system implementation

65 vulnerabilities in 12 applications

ALICE: A tool to find ALC vulnerabilities

BOB: A tool to determine persistence properties

Vulnerabilities follow patterns

Questions?