

The Wisconsin PASS Project

Andrea Arpaci-Dusseau,
Remzi Arpaci-Dusseau,
Ben Liblit, Miron Livny,
Michael Swift

University of Wisconsin, Madison

**Our Goal:
Develop Techniques
for Building Robust & Reliable
File Systems**

Current Efforts

Outline

I/O Shepherding [SOSP '07]

NTFS Study [StorageSS '06 + In Progress]

Error Propagation Analysis [In Progress]

Driver Refactoring [HotOS '07 + In Progress]

Latent Sector Study [Sigmetrics '07]

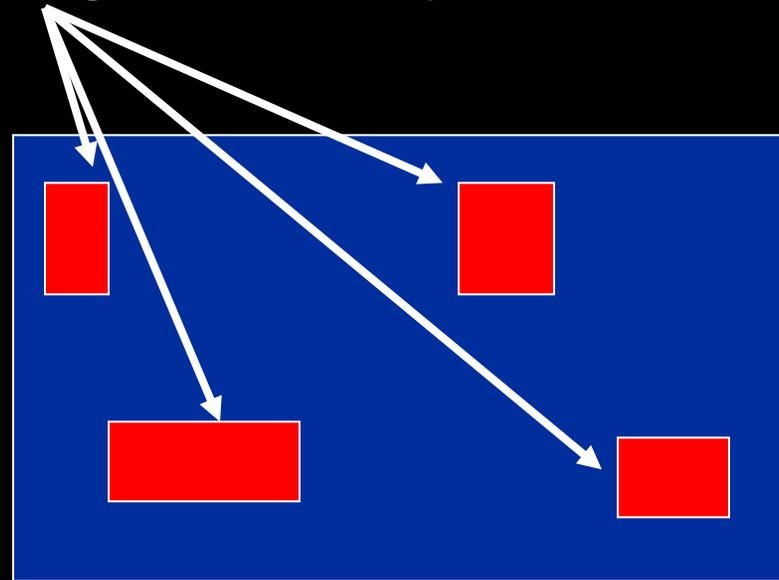
NFS Study [StorageSS '07]

I/O Shepherding

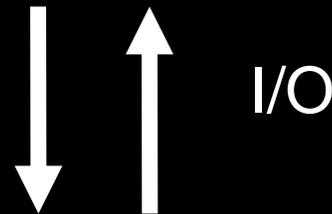
Haryadi Gunawi (Wisconsin),
Vijayan Prabhakaran (MSR),
Shweta Krishnan (Wisconsin),
Andrea Arpaci-Dusseau,
Remzi Arpaci-Dusseau
(Wisconsin)

Typical Reliability Features

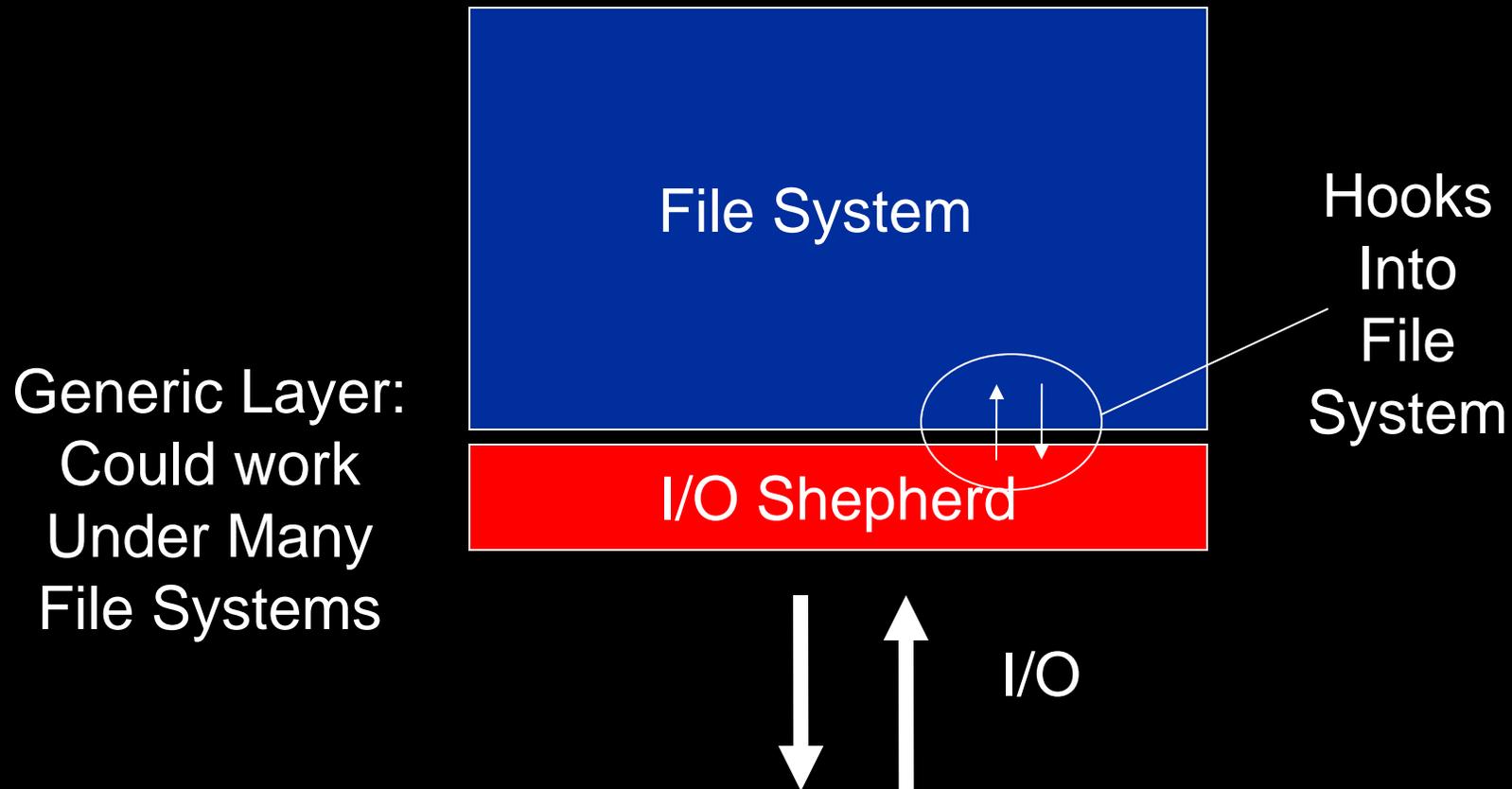
Fault Handling + Reliability Code



File System



I/O Shepherding



I/O Shepherd

Inputs

- An I/O operation on a particular block type
 - e.g., write inode X to address A
- Policy code
 - Specifies how I/O should be handled

Shepherd

- Responsible for “care and feeding” of I/O as dictated by policy code

Specifying Policy: Policy Code

Example code

- **ParanoidRead** (DiskAddr D, MemAddr M)
if (IOS_Read (D, M) == OK)
return OK;
else
IOS_Stop(IOS_HALT); // primitive
- **RetryPropagateRead** (DiskAddr D, MemAddr M)
for (int i = 0; i < RETRY_MAX; i++)
if (IOS_Read (D, M) == OK)
return OK;
return FAILURE;

Code segment **per block type**

- Enables fine-grained policies
(e.g., different treatment for metadata, data)

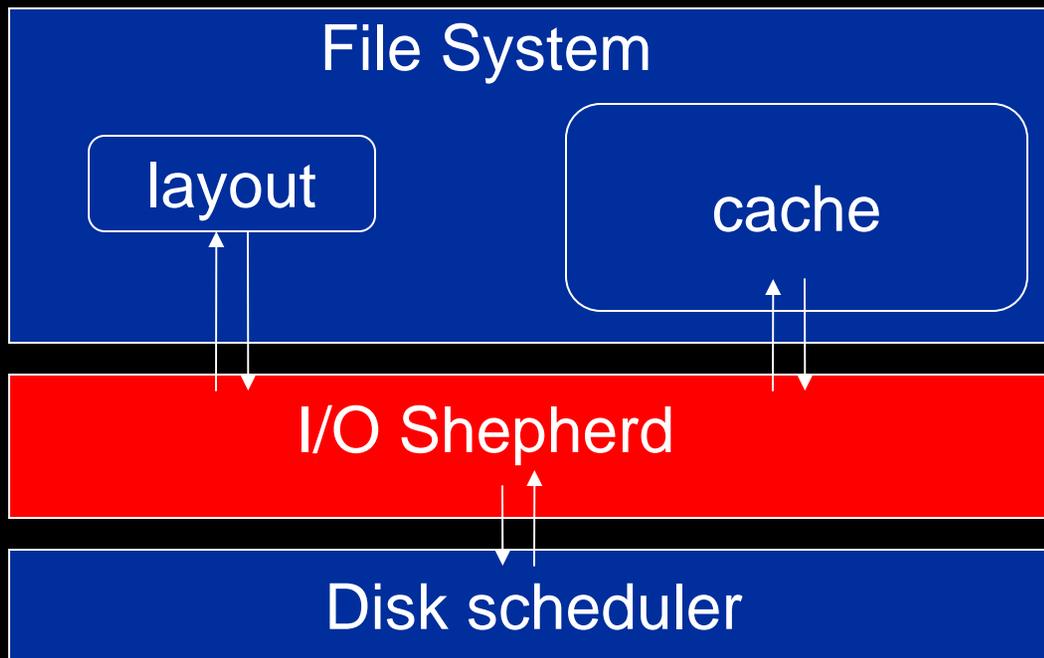
Shepherd: Not A Simple Layer

Loss of Information:

- Used to know lots of info about each request
- Shepherd: Add type info to each request as it flows thru system

File layout:

- Policy wants to make copy of a block
- Interface to FS layout and allocation engine



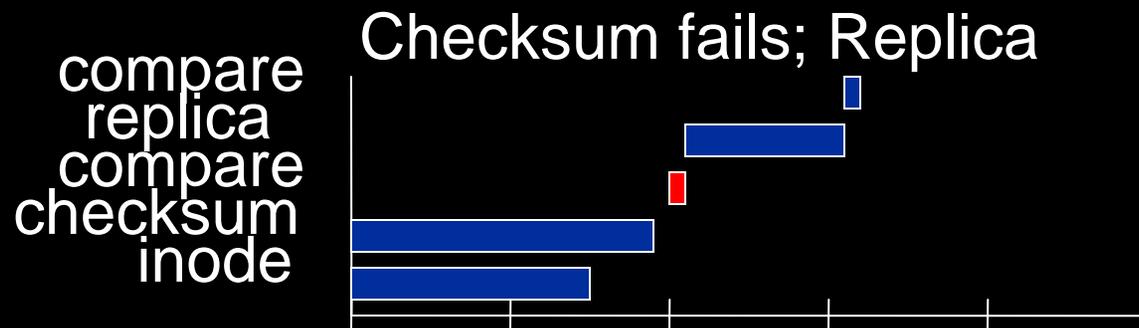
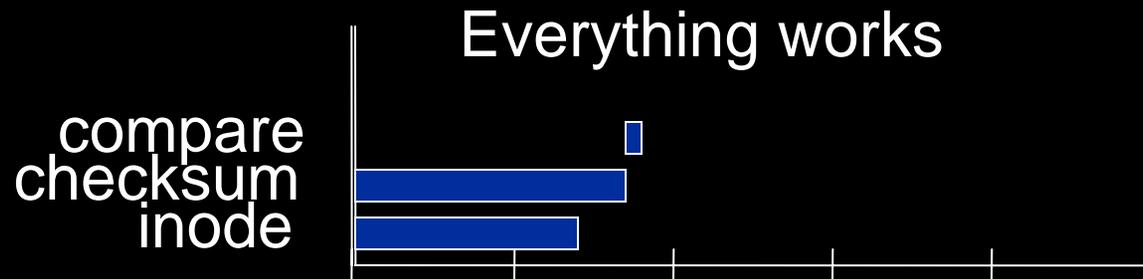
Disk scheduling:

- Policy wants to read a block or replica
- Disk scheduler must know possible locations

Caching:

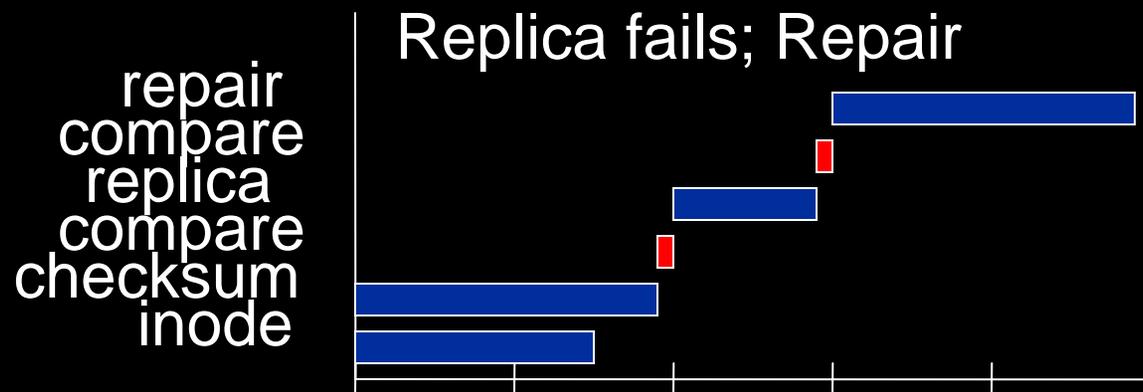
- Don't pollute cache with multiple copies
- Shepherd must interact with cache properly

A Multi-Level Policy



Policy

- Checksum
- Replica
- Semantic repair



Time (ms)

Shepherding Summary

Shepherding goal

- Making reliability a first-class FS concern

Which boils down to

- Simple description of powerful policies
- Efficient implementation

Examples (not shown here):

- Adding checksums, mirrors, parity, stronger sanity checks
- All well integrated with rest of system (little overhead)

NTFS Study

Lakshmi Bairavasundaram,
Meenali Rungta,
Andrea Arpaci-Dusseau,
Remzi Arpaci-Dusseau, Mike Swift
(Wisconsin)

The Question

How does a commercial file system react to corrupt data?

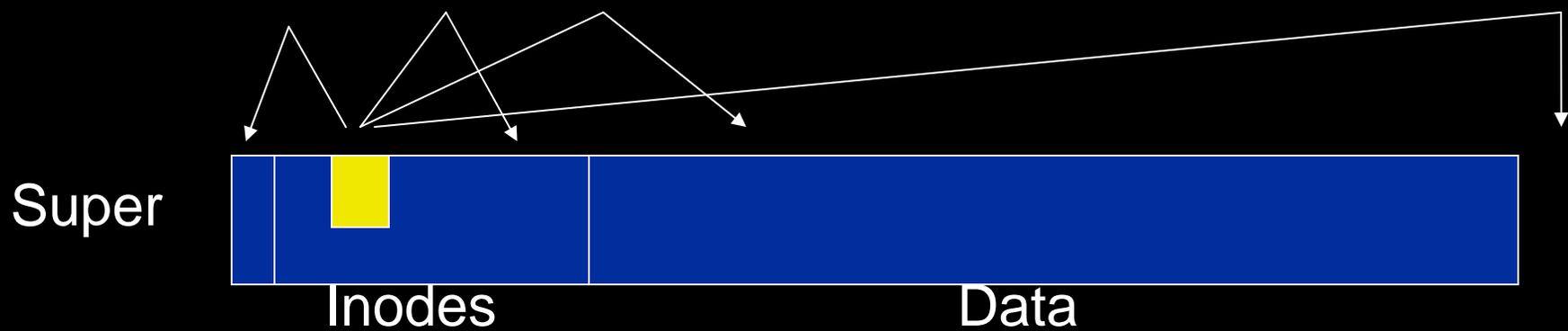
Case study: NTFS

- Modern file system
- Great deal of internal type information
- Also, some FS structure replication

Technique: Pointer Corruption

Fault injection technique: Pointer corruption

- Modify each on-disk pointer to point to each other type of structure
- Observe reaction of file system



Results: Qualitative

Type checking is used frequently

- But often not enough (type overloading)
- Does not work for all types (e.g., FileData)

Limited sanity checks

- Not very consistent

Replication used to recover from corruption

- But sometimes propagates corruption
- And can't tell difference between target corruption and pointer corruption

Loss of performance-aiding structures
is catastrophic (could be rebuilt instead)

Error Propagation

Haryadi Gunawi, Cindy Rubio,
Andrea Arpaci-Dusseau,
Remzi Arpaci-Dusseau, Ben Liblit
(Wisconsin)

The Problem

Problem: Lost Errors

- Low-level generates error (EIO)
- Somewhere before it gets reported, file system loses the error

Causes many problems

- Can't tell if operation worked
- File system itself can't detect/recover

How to find where these occur?

The Approach

Static source code analysis

- Look through source code for places where error codes are lost

Built in CIL framework (from UCB)

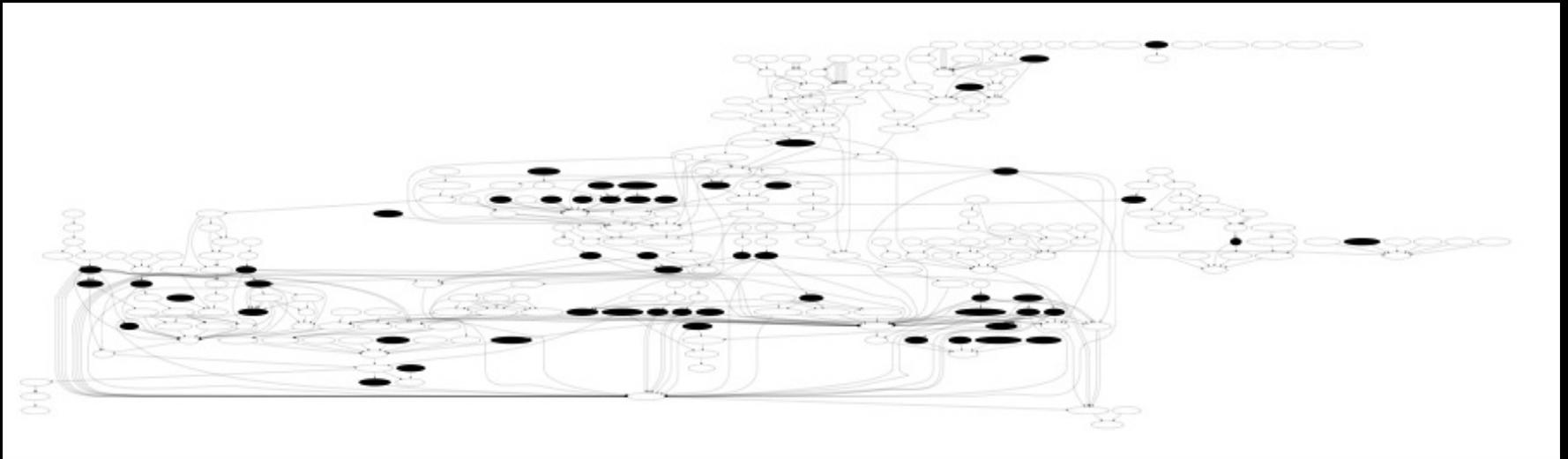
- Error generation: Return value or arg
- Constructs channels: Flow of errors back through call graph
- Determine if channel is “error complete”, *i.e.*, logs error, or takes recovery action
- All other channels marked “broken”

Results: Example

```
int sync_blockdev (block_device *) {  
    int ret = 0, err;  
    ret = filemap_fdatewrite();  
    err = filemap_fdatawait();  
    if (!ret) {  
        ret = err;  
        return ret;  
    }  
}
```

```
int journal_recover (journal *) {  
    int err;  
    ...  
    sync_blockdev(); // ERROR IGNORED!  
    ...  
    return err;  
}
```

Overall Results



Results

- Linux ext2, ext3, JFS, ReiserFS
- Numerous paths where errors are lost (over 90 in ext3 version we analyzed)

Automatic Driver Refactoring

Vinod Ganapathy,
Arini Balakrishnan,
Somesh Jha, Mike Swift
(Wisconsin)

The Problem

Drivers: Major cause of failure in systems

- 85% of failures in Windows XP

Why so bad?

- Fundamentally hard code to get right
 - Subtle kernel interfaces
 - Concurrency
 - Locking

How can we improve this situation?

The Solution

Assumption

- The less kernel code, the better

Microdrivers: Split drivers in two

- Kernel: performance critical
- Usermode: all other infrequently used parts
(e.g., configuration, error handling)

But, how to achieve the split?

Automatic Splitting

Splitter:

- Analyze driver
- Identify “critical root functions” (CRFs)
 - Interrupt handling, data transfer
- Also, all routines that are called by CRFs
- Mark all other code as non-critical

Code generator:

- Generate split kernel/user code
- Generate glue code for communication

Preliminary Results

Code removal: How much code can be moved to user space?

- Network: 72%
- SCSI: 74%
- Sound: 92%

Performance: How is performance affected by the split?

- Network: 6% overhead in worst case

Future Directions

Future Directions

Model-Checked Shepherd Policies

- Basic failure policies in place
- Does policy work as intended?
(can we **prove** that it does?)
- Can we use computational resources to aid checking? (Condor)

Improved Error Propagation Analysis

- Currently somewhat primitive
- Problems: Asynchronous I/O, complex transformations

Future Work

Analyze more file systems

- Sun ZFS
- NFS (Linux implementation)
- A cluster file system (still thinking about which one ...)

Microdrivers

- Full implementation
- More drivers
- Performance analysis

Future Work

Reliability Analysis of WAFL (RAW)

- NetApp Write-Anywhere File System (WAFL)
- 15 years of development
- Can we describe its fault-handling policy, find flaws, etc.?

Corruption Study

- NetApp failure data
- How often do disks return corrupted data?

People

Current People (Students)

Lakshmi Bairavasundaram (Ph.D.)

- Disk study, NTFS, Corruption

Meenali Rungta (M.S., Google)

- NTFS

Haryadi Gunawi (Ph.D.)

- CFM, Error propagation

Shweta Krishnan (M.S., Cisco)

- CFM

Arini Balakrishnan (M.S., Sun)

- Microdrivers

Cindy Rubio (Ph.D.)

- Error propagation

Andrew Krioukov (Undergraduate)

- RAW

Questions?

www.cs.wisc.edu/adsl