# From Consistency Checking to Journaling

Azza Abouzied

Critique due on Thursday (1-2 page) on The Design and Implementation of a Log-Structured File System
by Mendel Rosenblum and John K. Ousterhout
in SOSP 1991

It is a seminal paper! So lots of positive things to say :)

We will go over it in detail next class.

# Reading a Paper & Writing a Critique

How to read a paper in general?

## The Design Arguments

Evidence
How do you know?

**Need Thesis**

*Stakeholders + Domain* — Entity ***E***, in context ***C***
*Core Tension* — with goal ***G*** but obstacles ***O***

Evidence
How do existing approaches fail ?

Satisfy constraints ***X***
Minimize costs ***Y***
Avoid obstacles ***Z***

*Axioms* — Principles & Constraints ***A***

**Approach Thesis**

Has characteristics ***T***
Achieve goal ***G***
Avoids obstacles ***O***

What did they borrow from other solutions in other settings?

What makes this solution different from those that failed?

What was the response to the proposed approach?

# FFS Annotated

## A Fast File System for UNIX*

*Marshall Kirk McKusick, William N. Joy†,*
*Samuel J. Leffler‡, Robert S. Fabry*

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720

*ABSTRACT*

A reimplementation of the UNIX file system is described. The reimplementation provides substantially higher throughput rates by using more flexible allocation policies that allow better locality of reference and can be adapted to a wide range of peripheral and processor characteristics. The new file system clusters data that is sequentially accessed and provides two block sizes to allow fast access to large files while not wasting large amounts of space for small files. File access rates of up to ten times faster than the traditional UNIX file system are experienced. Long needed enhancements to the programmers' interface are discussed. These include a mechanism to place advisory locks on files, extensions of the name space across file systems, the ability to use long file names, and provisions for administrative control of resource usage.

**Annotate the paper with the design arguments: Goals, Characteristics, Approach, Context, Needs, … etc.**

# Writing a Critique

Read the blog post https://azzablogs.com/2019/01/23/how-to-write-a-critique-for-a-research-paper/

*When writing a summary of the paper, use the design arguments to help your describe the work.*

**Great critiques anticipate future contexts and needs and re-examine the work in light of those.**

1. Discuss Pros and Cons with the future in mind (or the present if the paper is from the past).
2. Does the work handle new technology (e.g. hardware, applications) in the horizon?
3. Does it scale to current workloads (consumer or enterprise)?
4. If not, why and how would you modify it?

# What makes a FS inconsistent?

# What are inconsistencies?

1. Same block in multiple files
2. Free blocks not in free list
3. Directory pointers to nowhere
4. Orphans
5. Funny attributes: bizarre modification times

A consistent file system might still have corrupt data

# How do consistency checkers work?

**Checkers:**
Unix: fsck; Windows: chkdisk, scandisk

1. Start from root (/) inode
2. Traverse the entire directory tree and mark reachable files/blocks
3. Verify the logical structure
4. Figure out which blocks are free

Do garbage collection: put free blocks in free list
Put orphaned files in /lost+found

# Unix (ext2) Consistency

Synchronous write-through for meta-data

Multiple updates are performed in a specific order

**When a crash occurs:**

1. Scan disk for consistency
2. Check for in-progress operations and fix up problems such as file created but not in directory, block allocated but not reflected in bitmap, etc.

What about data consistency?

# Unix (ext2) Consistency

Synchronous write-through for meta-data $\rightarrow$ poor performance

Multiple updates are performed in a specific order

**When a crash occurs:** $\rightarrow$ slow recovery!

1. Scan disk for consistency
2. Check for in-progress operations and fix up problems such as file created but not in directory, block allocated but not reflected in bitmap, etc.

What about data consistency? $\rightarrow$ A flush every 30 sec!

# Order Example

Suppose you want to extend the file by one block.

First **find a free block**, then the set of write operations (in no particular order) are:

- write data
- write block bitmap
- write inode with pointer to free block and new file size

**What if one write of the three succeeds?**

- Just the data block?
- Just the updated inode is written to disk?
- Just the updated bitmap is written to disk?

# Order Example

Suppose you want to extend the file by one block.

First **find a free block**, then the set of write operations (in no particular order) are:

- write data
- write block bitmap
- write inode with pointer to free block and new file size

**What if one write of the three succeeds?**

- Just the data block? $\rightarrow$ It is as if nothing happened! Data is written but no way to get to it.
- Just the updated inode is written to disk? $\rightarrow$ inode pointer to garbage and block appears free.
- Just the updated bitmap is written to disk? $\rightarrow$ No idea who owns the no longer free block

# Order Example

First **find a free block**, then the set of write operations (in no particular order) are:

- write data
- write block bitmap
- write inode with pointer to free block and new file size

**What if two writes succeed?**

- Inode and data bitmap updates succeed?
- Inode and data block updates succeed?
- Data bitmap and data block succeed?

# Order Example

What order should you perform the write operations?

# Consistent updates

**Expanding a file by a block**

Metadata first

1. Find a data block
2. Write pointer into i-node
3. Write new data to the data block
4. update free bitmap

Data first

1. Find a data block
2. Write new data to block
3. Write pointer into i-node
4. Update free bitmap

Consistent updates work by ensuring a certain order of instructions.

# Consistent Updates

Creating a new file

1. Write data block
2. Update inode
3. Update inode bitmap
4. Update free bitmap
5. Update directory

*If directory needs another data block:*

1. update free bitmap
2. update directory inode

# Consistent Updates

**Creating a new file**

1. Write data block CRASH $\rightarrow$ writes disappear: do nothing
2. Update inode
3. Update inode bitmap
4. Update free bitmap
5. Update directory

*If directory needs another data block:*

1. update free bitmap
2. update directory inode

# Consistent Updates

Creating a new file

1. Write data block
2. Update inode
3. Update inode bitmap
4. Update free bitmap
5. Update directory CRASH $\rightarrow$ File created but not in any directory: delete file/move to lost+found

*If directory needs another data block:*

1. update free bitmap
2. update directory inode

# Can you get better data consistency?

**Sometimes meta-data consistency is good enough**

How should vi save changes to a file to disk?

1. Write new version in temp file
2. Move old version to another temp file
3. Move new version into real file
4. Unlink old version

If crash, look at temp area; if any files out there, notify user that there might be a problem!

# Transactions

# From Consistent Updates to Transactions

What if multiple file operations need to occur as a unit: money transfer for e.g? atomic operations?

*DB concept: group many operations into a transaction and ensure* ACID

1. Atomicity: the collection of txns either happens or it doesn't … no partially happened
2. Consistency: We move from one consistent state to the next
3. Isolation (Serializability): Transactions appear to happen one after the other
4. Durability (Persistence): once it happens, it happened (no data loss)

# Money Transfer Example

Consider moving 100$ from account A to B

```
T1:
Begin
  If(A >= 100)
    A = A - 100
    B = B + 100
  else
    Abort
Commit
```

1. Transactions can run **concurrently** so we must ensure **Isolation** (What happens if T2 deletes A?)
2. **Aborts** (also crashes) can happen anytime

# Ensuring Isolation

Simple Locking scheme: **Two phase locking (2PL)**
Two lock types: shared/read locks & exclusive/write locks.

Phase 1 (Growing): Acquire all locks on records that a tx will affect
You can upgrade a read to a write lock (but you can't downgrade)

Phase 2 (Shrinking): Release locks cannot acquire new ones.

**Commit or Abort.**

*Does this ensure absence of deadlocks?*

*Does this ensure isolation?*

# Everything Else

*Critical sections give us atomicity and serializability, but not durability*

**Write Ahead Logging**

Begin Log all updates to a Write-Ahead-Log on disk

Commit

1. Write commit to the end of the log.
2. Then actually write the updates to the correct locations on disk
3. Clear the log

Abort Clear the log

**Crash Recovery**
*No commit:* do nothing
*Commit:* replay the log, then clear the log

# WAL in Action

```
Begin
    A = A - 100
    B = B + 100
Commit
```

1. Write A-=100 to log
2. Write B+=100 to log
3. Write commit to log
4. Write A to disk
5. Write B to disk
6. Clear the log

Can we swap 3 and 4? Can we swap 4 and 5?

# Transactions and File Systems

**Option 1: Each operation is a transaction**

Create/move/write

Does this eliminate the need for fsck?

**Option 2: Arbitrary # of operations form a transaction**

Log operations to make a very long operation

**Recovery**: replay the log

*This is* **journaling or logging** *file system: Windows NTFS, Mac OS X Extended, Linux ext3 File system, etc.*

1. Write an entry in the journal to describe the change
2. Implement the change in the file system
3. Mark the journal entry as completed
4. Eventually reclaim space used by completed journal entries

# Journaling and WAL

**What happens on a crash?**

At recovery time, instead of scanning the entire disk to look for inconsistencies, we can just look at the uncompleted journal entries and carry out whichever ones have not actually taken effect in the main filesystem. (Any partial journal entries are ignored.) Linux ext3 filesystem.

**What happens if we journal adding a block to the end of a file before we actually write the block?**

Some of this corruption can be avoided by carefully scheduling the order of disk operations, but this may conflict with other disk scheduling goals (like not moving the head too much). Ext3 has barriers that force order!

# Journaling Key Advantage

**Guarantee consistency with minimal recovery time**

**High price:** We update twice, once in the journal and once in the actual file. But we can

1. do actual writes later
2. sequential log means higher write bandwidth
3. Use Flash or NVRAM for logs
4. **Journal metadata only and not file contents**

# What to Log?

**Physical**

Log block images

Before: enables rollback

After: enables moving forward

Both: go either way

**Logical**

Example: Add file x to directory y

More compact more recovery work!

Questions?