



# The Write-Anywhere-File- Layout (WAFL)

Ohad Rodeh

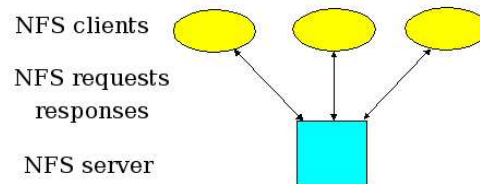


# Introduction

- This lecture is based on **File System Design for an NFS File Server Appliance** Dave Hitz, James Lau, Michael Malcolm. Proceedings of the USENIX Winter 1994 Technical Conference
- The WAFL design is used today in Network-Appliance filers

# Appliance for NFS

- NFS is Network File System
  1. A standard protocol to access a remote file-system
  2. Create/Delete file
  3. Read/Write
- An appliance is a special purpose device





# Main ideas

---

- New file-system idea
  1. Write-Anywhere-File-Layout (WAFL)
  2. Create snapshots efficiently
  3. Uses a copy-on-write technique
  4. Minimizes disk-space that snapshots consume
- Snapshots are used to eliminate the need for file-system consistency checks after an unclean shutdown



# Design goals

- WAFL is to be used inside an NFS appliance
- Tuned specifically for NFS
- Increased write workload due to problematic caching at client
- WAFL should:
  1. Provide fast NFS service
  2. Support large file systems (tens of GB) that grow dynamically as disks are added. This simplifies management, users don't want multiple partitions.
  3. Provide high performance while supporting RAID
  4. Support RAID in order to tolerate failed disks gracefully



# Snapshots

- Snapshot: read-only copy of the entire file system.
- Why use snapshots?
  1. Users can access snapshots through NFS to recover files that they have accidentally changed or removed
  2. System administrators can use snapshots to create backups safely from a running system
  3. WAFL uses Snapshots internally so that it can recover quickly from crashes

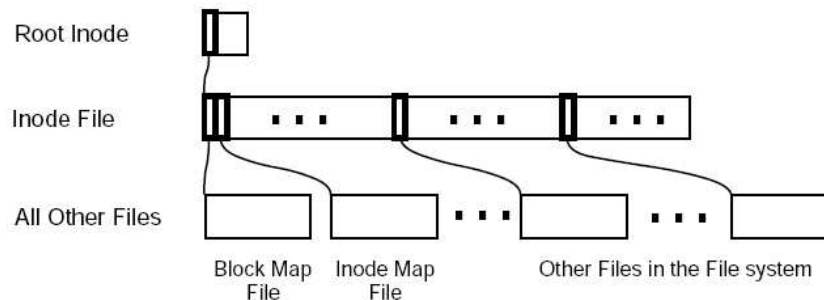


# Structure

- WAFL is similar in many ways to FFS
- It uses 4 KB blocks with no fragments
- The i-node is similar to FFS, with some exceptions
  1. Contains 16 block pointers
  2. All the block pointers refer to blocks at the same level
  3. I-nodes for files smaller than 64 KB use the 16 block pointers to point to data blocks
  4. I-nodes for files larger than 64 KB point to indirect blocks which point to actual file data
  5. I-nodes for even larger files point to doubly indirect blocks

# Meta data files

- WAFL stores meta-data in files
- There are three meta-data files
  1. I-node file, contains the i-nodes for the file system
  2. Block-map file, identifies free blocks
  3. I-node-map file, identifies free i-nodes
- The term map is used instead of bit map because these files use more than one bit for each entry





# Why use files to hold meta-data?

- Allows writing meta-data blocks anywhere on disk
- Write-anywhere allows operating efficiently with RAID
  1. WAFL works with RAID-4
  2. Scheduling multiple writes to the same RAID stripe whenever possible
  3. Avoid the 4-to-1 short-write penalty
- Makes it easy to increase the size of the file system on the fly

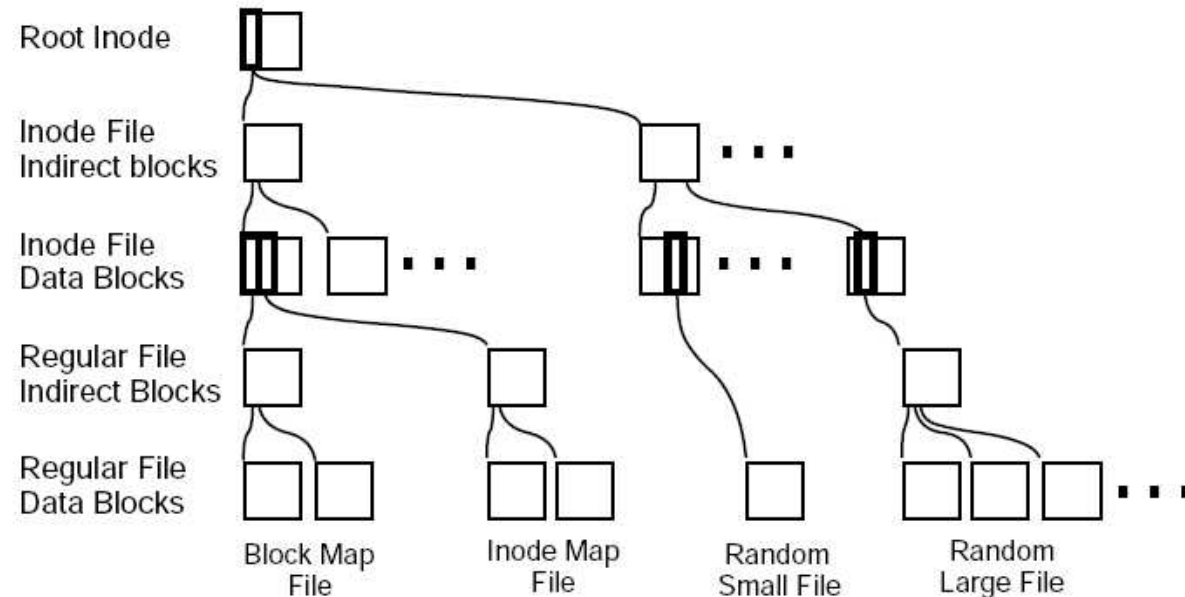


# Why use files to hold meta-data? II

- When a new disk is added
  1. Server automatically increase the sizes of the meta-data files
  2. The system administrator can increase the number of i-nodes in the file system manually if the default is too small

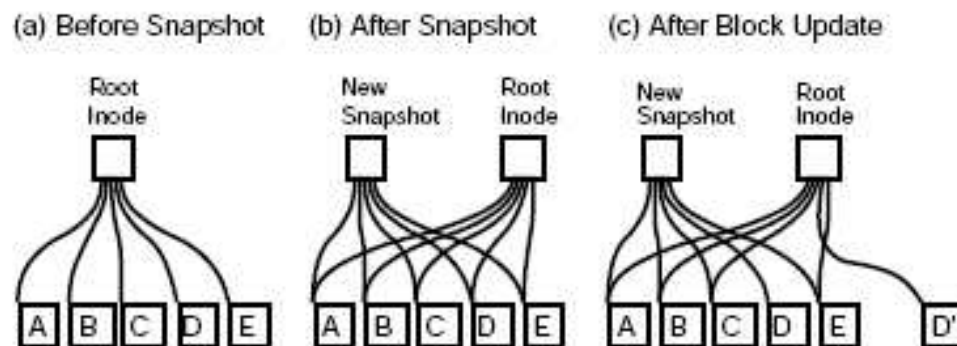
# Tree of blocks

- A WAFL file system is best thought of as a tree of blocks



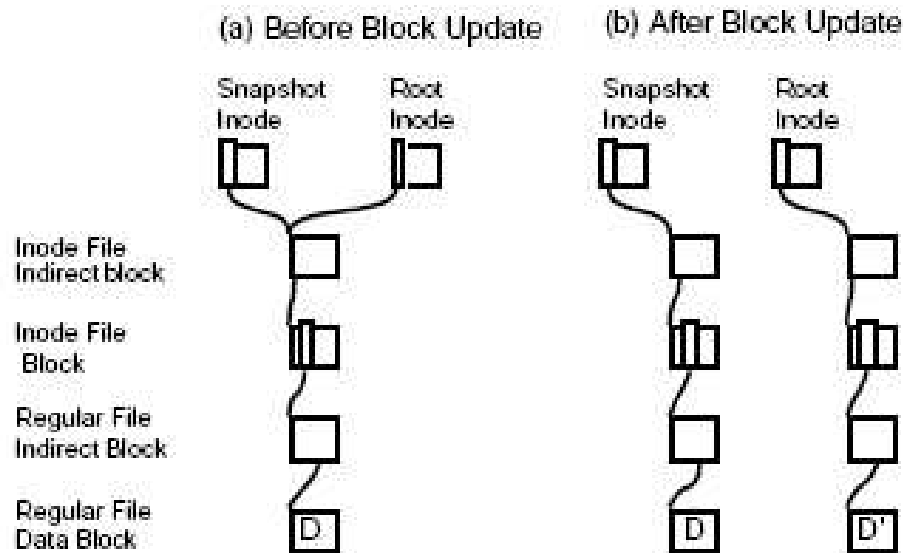
# Creating a snapshot

- WAFL creates a Snapshot by duplicating the root i-node that describes the i-node file
- WAFL avoids changing blocks in a Snapshot by writing new data to new locations on disk
- Snapshot creation is very quick



# Updates propagate up the tree

- To write a block to a new location, the pointers in the block's ancestors must be updated, which requires them to be written to new locations as well.





# Coalescing block modifications

- Each NFS request causes modifications to many blocks
- WAFL would be very inefficient if it wrote this many blocks for each NFS write request.
- Instead, gather up many hundreds of NFS requests before scheduling a write episode.
- During a write episode:
  1. Allocate disk space for all the dirty data in the cache
  2. Schedule the required disk I/O
  3. Commonly modified blocks, such as indirect blocks and blocks in the i-node file, are written only once per write episode instead of once per NFS request



# Consistency and recovery

- WAFL creates checkpoints once every 10 seconds
- A checkpoint is a snapshot that is not accessible to users
- Between checkpoints:
  1. NFS requests are recorded in the log
  2. The log is on NVRAM, so access is fast
  3. Blocks that are in use are never overwritten
  4. Ensures that the previous checkpoint remains unchanged



# Consistency and recovery II

---

- In case of crash:
  1. Go to latest checkpoint
  2. Replay the log
- File system state advances atomically between checkpoints



# Continued operation at checkpoint

- WAFL divides the NVRAM into two separate logs
- When one log gets full, WAFL switches to the other log and starts writing a consistency point to store the changes from the first log safely on disk.
- Scheduling a consistency point every 10 seconds should, in most cases, avoid overflowing the log

# Why use logical logging

- Logging NFS requests to NVRAM is better than using NVRAM to cache writes at the disk driver layer
- Processing an NFS request and caching the resulting disk writes generally takes much more NVRAM than simply logging the information required to replay the request.
- Examples:
  - Write 4KB to a file, changes:
    1. File-attributes (mtime)
    2. Data block
    3. Indirect-block
  - Rename



# Why use logical logging II

- NVRAM also allows quick response to NFS requests
- With a typical mix of NFS operations, WAFL can store more than 1000 operations per megabyte of NVRAM.



# Why optimize for writes?

- Write performance is especially important for network file servers.
- As read caches get larger at both the client and server, writes begin to dominate the I/O subsystem
- This effect is especially pronounced with NFS which allows very little client-side write caching.
- Result: the disks on an NFS server may have 5 times as many write operations as reads



# Write allocation

- WAFL can write any file system block (except the one containing the root i-node) to any location on disk.
  1. In FFS, meta-data is kept at fixed locations
  2. Prevents FFS from optimizing writes by, for example, putting both the data for a newly updated file and its i-node right next to each other on disk



# Write allocation II

- WAFL can write blocks to disk in any order
  1. FFS writes blocks to disk in a carefully determined order so that fsck(8) can restore file system consistency after an unclean shutdown.
  2. WAFL's constraint: must write all the blocks in a new consistency point before it writes the root i-node for the consistency point.



# Delayed allocation

- WAFL gathers up hundreds of NFS requests before scheduling a consistency point
- Then it allocates blocks for all requests in the consistency point at once
- Deferring write allocation
  1. Improves the latency of NFS operations by removing disk allocation from the processing path of the reply
  2. Avoids wasting time allocating space for blocks that are removed before they reach disk



# Allocation heuristics

- Improve RAID performance by writing to multiple blocks in the same stripe
- Reduce seek time by writing blocks to locations that are near each other on disk
- Reduce head-contention when reading large files by placing sequential blocks in a file on a single disk in the RAID array



# Allocation map

---

- Most file systems
  1. Keep track of free blocks using a bit map with one bit per disk block
  2. If the bit is set, then the block is in use
- This technique does not work for WAFL because many snapshots can reference a block at the same time



# Allocation map II

- WAFL:

1. block-map file contains a 32-bit entry for each 4 KB disk block
2. Bit 0 is set if the active file system references the block
3. Bit 1 is set if the first Snapshot references the block, etc.
4. A block is in use if any of the bits in its block-map entry are set

# Example lifetime of a block

Time	Block-Map Entry	Description
t1	0 0 0 0 0 0 0 0	Block is unused.
t2	0 0 0 0 0 0 0 1	Block is allocated for active FS
t3	0 0 0 0 0 0 1 1	Snapshot #1 is created
t4	0 0 0 0 0 1 1 1	Snapshot #2 is created
t5	0 0 0 0 0 1 1 0	Block is deleted from active FS
t6	0 0 0 0 0 1 1 0	Snapshot #3 is created
t7	0 0 0 0 0 1 0 0	Snapshot #1 is deleted
t8	0 0 0 0 0 0 0 0	Snapshot #2 is deleted; block is unused

bit 0: set for active file system  
bit 1: set for Snapshot #1  
bit 2: set for Snapshot #2  
bit 3: set for Snapshot #3



# Create a snapshot

- The challenge: Avoid locking out incoming NFS requests.
- New NFS requests may need to change cached data that is part of the Snapshot and which must remain unchanged until it reaches disk
- The trivial solution:
  1. Suspend NFS processing
  2. Write the Snapshot
  3. Resume NFS processing
- However, writing a Snapshot can take over a second
- Too long for an NFS server to stop responding



# Keeping snapshot data self consistent

- Mark all the dirty data in the cache as IN\_SNAPSHOT
- During Snapshot creation:
  1. Data marked IN\_SNAPSHOT must not be modified
  2. Data not marked IN\_SNAPSHOT must not be flushed to disk



# Keeping snapshot data self consistent II

- NFS requests can
  1. Read all file system data
  2. Modify data that isn't IN\_SNAPSHOT
- Processing for requests that need to modify IN\_SNAPSHOT data must be deferred
- To minimize the delay for these requests, IN\_SNAPSHOT data must be flushed as quickly as possible

# Flushing IN\_SNAPSHOT data fast

- Allocate disk space for all files with IN\_SNAPSHOT blocks
  1. I-nodes are cached in
    - (a) A special in-core i-node cache
    - (b) Disk-buffers
  2. After allocation, copy i-nodes from in-core cache to disk buffers
  3. Allows continued operation on in-core i-nodes
  4. Does not require actual IO
- Update the block-map file: for each block-map entry copy the bit for the active file system to the bit for the new Snapshot

# Flushing IN\_SNAPSHOT II

- Write all IN\_SNAPSHOT disk buffers in cache to their newly-allocated locations on disk.
- As soon as a particular buffer is flushed, any NFS requests waiting to modify it can be restarted.
- Duplicate the root i-node
  1. Create an i-node that represents the new Snapshot
  2. Turn the root i-node's IN\_SNAPSHOT bit off
  3. The new Snapshot i-node must not reach disk until after all other blocks in the Snapshot have been written



# Creating a snapshot

- Once the new Snapshot i-node has been written
  1. No more IN\_SNAPSHOT data exists in cache
  2. Any NFS requests that are still suspended can be processed
- Under normal loads the four steps can be performed in less than a second.
- Step (1) can generally be done in just a few hundredths of a second, and once WAFL completes it, very few NFS operations need to be delayed

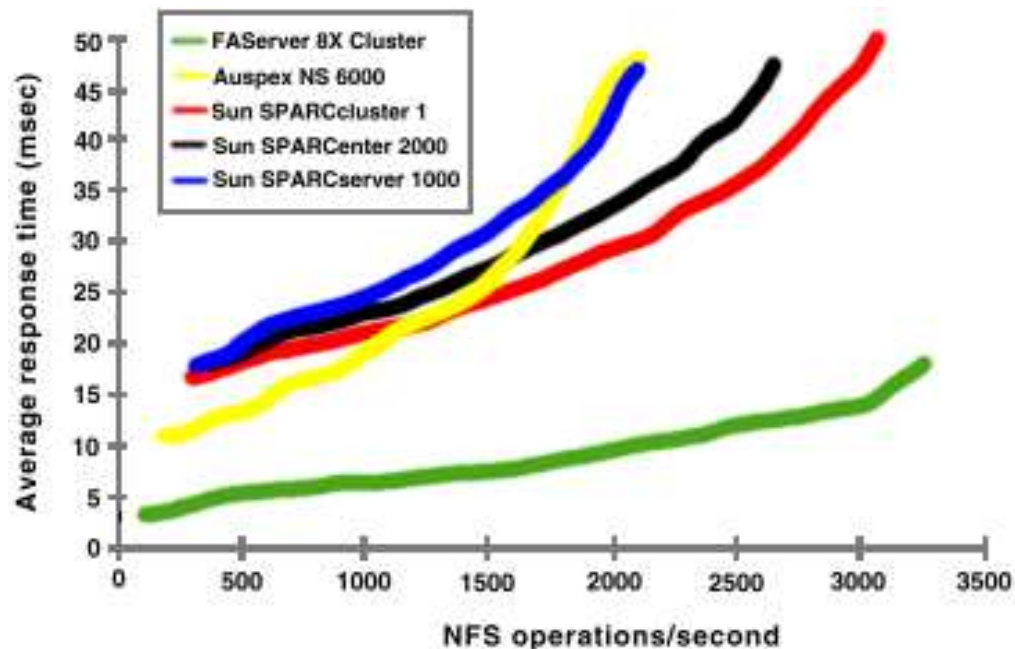


# Deleting a snapshot

- Deleting a Snapshot is trivial
  1. Zero the root i-node representing the Snapshot
  2. Clear the bit representing the Snapshot in each block-map entry

# Performance

- Performance comparison of several file-system from 1994
- WAFL has good performance
  1. Uses less file-systems (8 here)
  2. Other file-systems are not optimized for NFS





# Summary

---

- WAFL is a very interesting file-system
  1. Log-structure without a cleaner
  2. Appliance philosophy
  3. Optimized for writes
  4. NVRAM for log
  5. Cheap snapshots
- WAFL has been successfully used by Network-Appliance for the past 10 years
- Has set the industry bar in
  1. NFS serving
  2. Snapshot performance and ease of use