File System Design for an NFS File Server Appliance

Dave Hitz, James Lau, and Michael Malcolm

Technical Report TR3002
NetApp
2002

http://www.netapp.com/us/library/white-papers/wp_3002.html

(At WPI: http://www.wpi.edu/Academics/CCC/Help/Unix/snapshots.html)

Introduction

- In general, *appliance* is device designed to perform specific function
- Distributed systems trend has been to use appliances instead of general purpose computers. Examples:
  - *routers* from Cisco and Avici
  - network *terminals*
  - network *printers*
- For files, not just another computer with your files, but new type of network appliance
  - → Network File System (NFS) file server

Introduction: NFS Appliance

- NFS File Server Appliances have different requirements than those of general purpose file system
  - NFS access patterns are different than local file access patterns
  - Large client-side caches result in fewer reads than writes
- Network Appliance Corporation uses Write Anywhere File Layout (WAFL) file system

Introduction: WAFL

- WAFL has 4 requirements
  - Fast NFS service
  - Support large file systems (10s of GB) that can grow (can add disks later)
  - Provide high performance writes and support Redundant Arrays of Inexpensive Disks (RAID)
  - Restart quickly, even after unclean shutdown
- NFS and RAID both strain write performance:
  - NFS server must respond after data is written
  - RAID must write parity bits also

WPI File System

- CCC machines have central, Network File System (NSF)
  - Have same home directory for `cccwork2`, `cccwork3`…
  - `/home` has 10,113 directories!
- Previously, Network File System support from NetApp WAFL
- Switched to EMC Celera NS-120
  - → similar features and protocol support
- Provide notion of "snapshot" of file system (next)

Outline

- Introduction (done)
- Snapshots : User Level (next)
- WAFL Implementation
- Snapshots: System Level
- Performance
- Conclusions

Introduction to Snapshots

- *Snapshots* are copy of file system at given point in time
- WAFL creates and deletes snapshots automatically at preset times
  - Up to 255 snapshots stored at once
- Uses *copy-on-write* to avoid duplicating blocks in the active file system
- Snapshot uses:
  - Users can recover accidentally deleted files
  - Sys admins can create backups from running system
  - System can restart quickly after unclean shutdown
    - Roll back to previous snapshot

User Access to Snapshots

- Example, suppose accidentally removed file named "todo":

```
CCCWORK3% ls -lut .snapshot/*/todo
-rw-rw---- 1 claypool claypool 4319 Oct 24 18:42
.snapshot/2011_10_26_18.15.29/todo
-rw-rw---- 1 claypool claypool 4319 Oct 24 18:42
.snapshot/2011_10_26_19.27.40/todo
-rw-rw---- 1 claypool claypool 4319 Oct 24 18:42
.snapshot/2011_10_26_19.37.10/todo
```

- Can then recover most recent version:

```
CCCWORK3% cp .snapshot/2011_10_26_19.37.10/todo todo
```

- Note, snapshot directories (`.snapshot`) are hidden in that they don't show up with `ls` (even `ls -a`) unless specifically requested

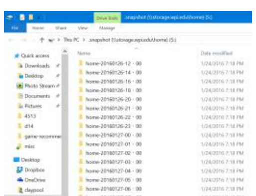
Snapshot Administration

- WAFL server allows sys admins to create and delete snapshots, but usually automatic
- At WPI, snapshots of `/home`. Says:
  - 3am, 6am, 9am, noon, 3pm, 6pm, 9pm, midnight
  - Nightly snapshot at midnight every day
  - Weekly snapshot is made on Saturday at midnight every week
  - → But looks like every 1 hour (fewer copies kept for older periods and 1 week ago max)

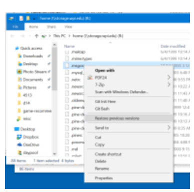
```
claypool 168 CCCWORK3% cd .snapshot
claypool 169 CCCWORK3% ls -1
home-20160121-00:00/
home-20160122-00:00/
home-20160122-22:00/
home-20160123-00:00/
home-20160123-02:00/
home-20160123-04:00/
home-20160123-06:00/
home-20160123-08:00/
home-20160123-10:00/
home-20160123-12:00/
…
home-20160127-16:00/
home-20160127-17:00/
home-20160127-18:00/
home-20160127-19:00/
home-20160127-20:00/
home-latest/
```

Snapshots at WPI (Windows)

- Mount UNIX space (`\\storage.wpi.edu\home`), add `\.snapshot` to end



Note, files in `.snapshot` do not count against quota



- Can also right-click on file and choose "restore previous version"

Outline

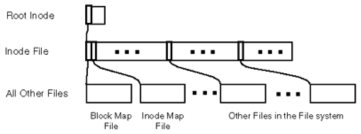
- Introduction (done)
- Snapshots : User Level (done)
- WAFL Implementation (next)
- Snapshots: System Level
- Performance
- Conclusions

WAFL File Descriptors

- Inode based system with 4 KB blocks
- Inode has 16 pointers, which vary in type depending upon file size
  - For files smaller than 64 KB:
    - Each pointer points to data block
  - For files larger than 64 KB:
    - Each pointer points to indirect block
  - For really large files:
    - Each pointer points to doubly-indirect block
- For very small files (less than 64 bytes), data kept in inode itself, instead of using pointers to blocks

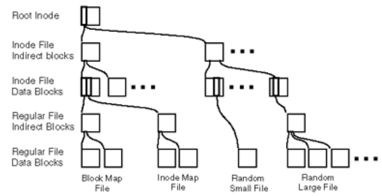
WAFL Meta-Data

- Meta-data stored in files
  - Inode file – stores inodes
  - Block-map file – stores free blocks
  - Inode-map file – identifies free inodes



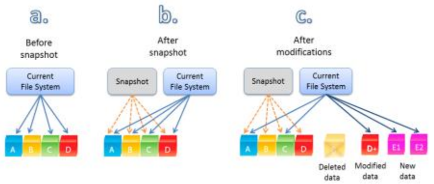
Zoom of WAFL Meta-Data (Tree of Blocks)

- Root inode must be in fixed location
- Other blocks can be written anywhere



Snapshots (1 of 2)

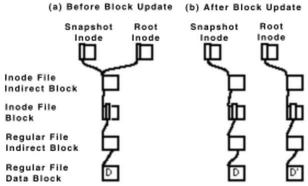
- Copy root inode only, copy on write for changed data blocks



- Over time, old snapshot references more and more data blocks that are not used
- Rate of file change determines how many snapshots can be stored on system

Snapshots (2 of 2)

- When disk block modified, must modify meta-data (indirect pointers) as well



- Batch, to improve I/O performance

Consistency Points (1 of 2)

- In order to avoid consistency checks after unclean shutdown, WAFL creates special snapshot called *consistency point* every few seconds
  - Not accessible via NFS
- Batched operations are written to disk each consistency point
  - Like journal
- In between consistency points, data only written to RAM

Consistency Points (2 of 2)

- WAFL uses NVRAM (NV = Non-Volatile):
  - (NVRAM is DRAM with batteries to avoid losing during unexpected poweroff, some servers now just solid-state or hybrid)
  - NFS requests are logged to NVRAM
  - Upon unclean shutdown, re-apply NFS requests to last consistency point
  - Upon clean shutdown, create consistency point and turnoff NVRAM until needed (to save power/batteries)
- Note, typical FS uses NVRAM for metadata write cache instead of just logs
  - Uses more NVRAM space (WAFL logs are smaller)
    - Ex: "rename" needs 32 KB, WAFL needs 150 bytes
    - Ex: write 8 KB needs 3 blocks (data, inode, indirect pointer), WAFL needs 1 block (data) plus 120 bytes for log
  - Slower response time for typical FS than for WAFL (although WAFL may be a bit slower upon restart)

Write Allocation

- Write times dominate NFS performance
  - Read caches at client are large
  - Up to 5*x* as many write operations as read operations at server
- WAFL batches write requests (e.g., at consistency points)
- WAFL allows "write anywhere", enabling inode next to data for better perf
  - Typical FS has inode information and free blocks at fixed location
- WAFL allows writes in any order since uses consistency points
  - Typical FS writes in fixed order to allow `fsck` to work if unclean shutdown

Outline

- Introduction (done)
- Snapshots : User Level (done)
- WAFL Implementation (done)
- Snapshots: System Level (next)
- Performance
- Conclusions

The Block-Map File

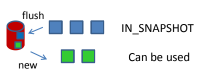
- Typical FS uses bit for each free block, 1 is allocated and 0 is free
  - Ineffective for WAFL since may be other snapshots that point to block
- WAFL uses 32 bits for each block
  - For each block, copy "active" bit over to snapshot bit

Time	Block-Map Entry	Description
t1	00000000	Block is unused
t2	00000001	Block is allocated for active FS
t3	00000011	Snapshot #1 is created
t4	00000111	Snapshot #2 is created
t5	00000110	Block is deleted from active FS
t6	00000110	Snapshot #3 is created
t7	00000100	Snapshot #1 is deleted
t8	00000000	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

Creating Snapshots

- Could suspend NFS, create snapshot, resume NFS
  - But can take up to 1 second
- Challenge: avoid locking out NFS requests
- WAFL marks all dirty cache data as IN_SNAPSHOT. Then:
  - NFS requests can read system data, write data not IN_SNAPSHOT
  - Data not IN_SNAPSHOT not flushed to disk
- Must flush IN_SNAPSHOT data as quickly as possible



Flushing IN_SNAPSHOT Data

- Flush inode data first
  - Keeps two caches for inode data, so can copy system cache to inode data file, unblocking most NFS requests
    - Quick, since requires no I/O since inode file flushed later
- Update block-map file
  - Copy active bit to snapshot bit
- Write all IN_SNAPSHOT data
  - Restart any blocked requests as soon as particular buffer flushed (don't wait for all to be flushed)
- Duplicate root inode and turn off IN_SNAPSHOT bit
- All done in less than 1 second, first step done in 100s of ms

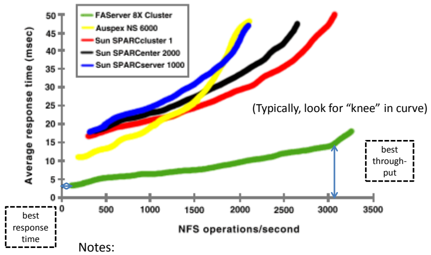
Outline

- Introduction (done)
- Snapshots : User Level (done)
- WAFL Implementation (done)
- Snapshots: System Level (done)
- Performance (next)
- Conclusions

Performance (1 of 2)

- Compare against other NFS systems
- How to measure NFS performance?
  - Best is SPEC NFS
    - LADDIS: Legato, Auspex, Digital, Data General, Interphase and Sun
- Measure response times versus throughput
  - Typically, servers quick at low throughput then response time increases as throughput requests increase
- (Me: System Specifications?!)

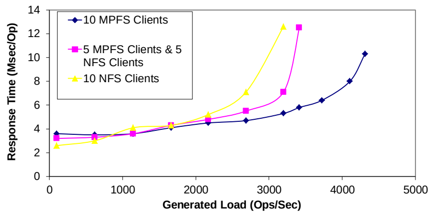
Performance (2 of 2)



Notes:
+ FAS has only 8 file systems, and others have dozens
- FAS tuned to NFS, others are general purpose

NFS vs. Newer File Systems

MPFS = multi-path file system
Used by EMC Celerra



- Remove NFS server as bottleneck
- Clients write directly to device

Conclusion

- NetApp (with WAFL) works and is stable
  - Consistency points simple, reducing bugs in code
  - Easier to develop stable code for network appliance than for general system
    - Few NFS client implementations and limited set of operations so can test thoroughly
- WPI bought one ☺