File System Design for and NSF File Server Appliance

Dave Hitz, James Lau, and Michael Malcolm

Technical Report TR3002
NetApp
2002

http://www.netapp.com/tech_library/3002.html

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

Introduction

- In general, an *appliance* is a device designed to perform a 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*
- New type of network appliance is an *Network File System (NFS) file server*

Introduction : NFS Appliance

- NFS File Server Appliance file systems have different requirements than those of a 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 a *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

Outline

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

Introduction to Snapshots

- *Snapshots* are a copy of the file system at a given point in time
  - WAFL's "claim to fame"
- 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":

```
spike% ls -lut .snapshot/*/todo
-rw-r--r-- 1 hitz 52880 Oct 15 00:00 .snapshot/nightly.0/todo
-rw-r--r-- 1 hitz 52880 Oct 14 19:00 .snapshot/hourly.0/todo
-rw-r--r-- 1 hitz 52829 Oct 14 15:00 .snapshot/hourly.1/todo
-rw-r--r-- 1 hitz 55059 Oct 10 00:00 .snapshot/nightly.4/todo
-rw-r--r-- 1 hitz 55059 Oct  9 00:00 .snapshot/nightly.5/todo
```

- Can then recover most recent version:

```
spike% cp .snapshot/hourly.0/todo todo
```

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

Snapshot Administration

- The WAFL server allows commands for sys admins to create and delete snapshots, but typically done automatically
- At WPI, snapshots of `/home`:
  - 7:00 AM, 10:00, 1:00 PM, 4:00, 7:00, 10:00, 1:00 AM
  - Nightly snapshot at midnight every day
  - Weekly snapshot is made on Sunday at midnight every week
- Thus, always have: 7 hourly, 7 daily snapshots, 2 weekly snapshots

```
claypool 32 ccc3=>>pwd
/home/claypool/.snapshot
claypool 33 ccc3=>>ls
hourly.0/  hourly.3/  hourly.6/  nightly.2/ nightly.5/ weekly.1/
hourly.1/  hourly.4/  nightly.0/ nightly.3/ nightly.6/
hourly.2/  hourly.5/  nightly.1/ nightly.4/ weekly.0/
```

Outline

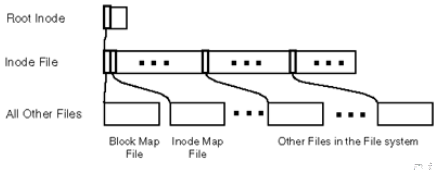
- Introduction (done)
- Snapshots : User Level (done)
- WAFL Implementation (next)
- Snapshots: System Level
- Performance
- Conclusions

WAFL File Descriptors

- Inode based system with 4 KB blocks
- Inode has 16 pointers
- 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 instead of pointers

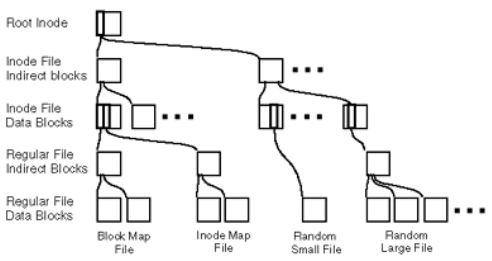
WAFL Meta-Data

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



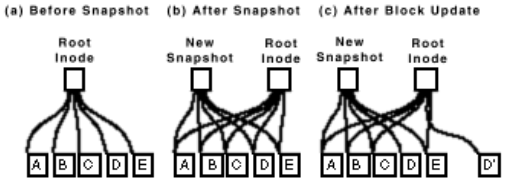
Zoom of WAFL Meta-Data (Tree of Blocks)

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



Snapshots (1 of 2)

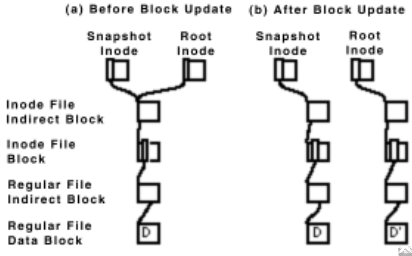
- Copy root inode only, copy on write for changed data blocks



- 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 the system

Snapshots (2 of 2)

- When disk block modified, must modify 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 a special snapshot called a *consistency point* every few seconds
  - Not accessible via NFS
- Batched operations are written to disk each consistency point
- In between consistency points, data only written to RAM

Consistency Points (2 of 2)

- WAFL use of NVRAM (NV = Non-Volatile):
  - (NVRAM has batteries to avoid losing during unexpected poweroff)
  - 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 batteries)
- Note, typical FS uses NVRAM for write cache
  - Uses more NVRAM space (WAFL logs are smaller)
    - Ex: "rename" needs 32 KB, WAFL needs 150 bytes
    - Ex: write 8KB 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

Write Allocation

- Write times dominate NFS performance
  - Read caches at client are large
  - 5x as many write operations as read operations at server
- WAFL batches write requests
- 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

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

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 (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
- Duplicate root inode and turn off IN_SNAPSHOT bit
- All done in less than 1 second, first step done in 100s of ms

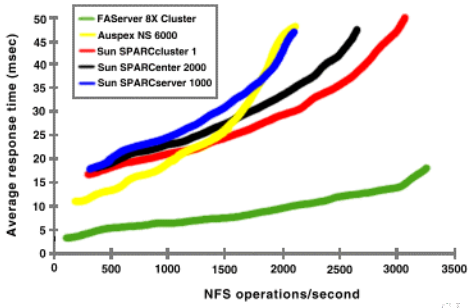
Outline

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

Performance (1 of 2)

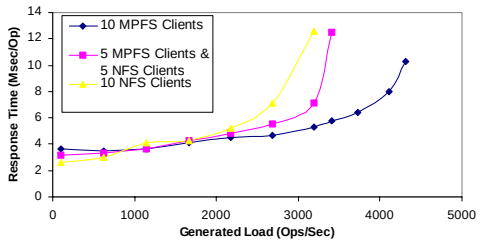
- Compare against NFS systems
- Best is SPEC NFS
  - LADDIS: Legato, Auspex, Digital, Data General, Interphase and Sun
- Measure response times versus throughput
- (Me: System Specifications?!)

Performance (2 of 2)



(Typically, look for "knee" in curve)

NFS vs. New File Systems



- 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 ☺