Storage Systems

NPTEL Course
Jan 2012
(Lecture 05)

K. Gopinath
Indian Institute of Science

Basic Storage API

- GetNew(oid)
    - POSIX: fd=creat(const char *path, mode_t mode)
- Store(oid, data)
    - POSIX: ssize_t write(int fd, const void *buf, size_t count)
- Read(oid, buffer)
    - POSIX: ssize_t read(int fd, void *buf, size_t count)
- Delete(oid)
    - POSIX: int unlink(const char *pathname)
- GetInfo(oid)
    - POSIX: int stat(const char *path, struct stat *buf)

oid: blocknum, filename, filehandle, content hash, key

Also, appl buffered versions.

Semantics of POSIX files vs NFS

- A fd once obtained can be held as long as the process exists (POSIX)
- A NFS handle once obtained can be held as long as the client exists
  - But may not be able to access the file if the NFS server dies
  - Server stateless but client stateful
- More interesting: open but deleted file
  - NFS can only approximate POSIX semantics
- Consistency guarantees weak in NFS2
  - Clients cache metadata for 30 secs

Impact of Networking

- Semantics of failure becomes imp
  - NFS uses RPC. What has to be done wrt non-idempotent operations?
- Consistency issues become important
  - NFS semantics different from POSIX
- To support high speed transfers, new kernel infrastructure
  - Parallel to IPC: sockets/TLI
  - For even higher speeds, user-level networking (RDMA)
- Storage spun off in large systems
  - Storage Area Networks (block level protocols)
  - NFS (file level protocols)
  - Distributed File Systems/Storage Systems

API changes

- POSIX: Read (fd, buffer, count)
  - Partial writes to a file OK (appends, overwrites, etc)
  - Mmap
- NFS: Read (fd, ***offset***, buffer, count)
  - Partial writes and mmap avlbl but no open!
  - Weak consistency model with multiple writers (NFS2)
    - NFS3, NFS4 improve the consistency model
- Amazon S3: “storage” service
  - Key Value store: no features like partial write or mmap
  - Weak consistency (“BASE”) model: when no updates occur for a “sufficiently long” period of time, eventually all updates will propagate through the system and all the replicas will be consistent.

S3 Interface: Key Value Store

- S3 stores data in named buckets
  - Each bucket is a flat namespace, containing keys associated with objects (but not another bucket)
  - Max obj size 5GB. Partial writes to objects not allowed (must be uploaded full), but partial reads OK
- create bucket
- put bucket, key, object
- get bucket, key
- delete bucket, key
- delete bucket
- list keys in bucket
- list all buckets

Layering in Storage Systems

- Varied uses of storage. Eg.
  - Swap
  - Document store
  - Archiving
  - Temporary info transfer (eg. Memory stick)
- Many designs. Best to understand each as a layered system with optional layers
  - Swap: no user visible component (block storage fine)
  - Document: metadata about document imp (provenance)
  - Archiving: reliability paramount and eliminating redundancy imp
- Simplified layering model: devices, protocols, systems

Storage Systems Highly Layered

- Multiple layers. Example:
  - Application uses fopen, fread, fwrite, etc.
  - Libc calls open, read, write system calls
  - Kernel calls vop_open, vop_read, vop_write, ...
  - FS implements ufs_open, ufs_read, etc. using virtual memory subsystem
  - Virtual Memory subsystem uses vop_getpage and vop_putpage provided by FS
  - vop_getpage/vop_putpage call pseudo device routines
  - Volume Manager or NFS client code
  - Device Driver (SCSI) or Network driver
  - HBA or NIC
  - Disk or Remote Disk

Layering

- Each layer often specializes in one dimension but has to handle others also
  - FS handles naming as reqd by appl
  - Volume Manager handles aggregation of physical media along with error mgmt
- Each layer also needs to do in its own way
  - Discovery
  - Naming
  - Error mgmt
  - Security
  - Performance (eg. Caching, Flow Control)
  - Consistency mgmt (transactions)

Let us start with the physical layer

- Disks: electromechanical devices
  - Dominant since 1956
  - Mostly replaced tape
  - May get replaced by storage class memory (SCM)
- High density with good BW but high seek and rotational delays
- Acceptable reliability but for large storage systems a big issue
  - Heat, power, vibration, ...
- Most software (fs, db, etc) till today optimized for disks

Disk Drive Interfaces

- Early disks: host just sees r/w amplifier (analog)
  - only soft sectoring
- ESDI disks: Only data separator
  - generates a clock and data signal from pulses in medium
  - hard sectoring; protocol with cmds; defect lists in drive
- SCSI disks: Also formatter, data buffer, controller
  - Most mature for large systems
- IDE/ATA disks: Also Host adapter in drive
  - disadv: only works with IBM PC
- SATA, SAS: serial ATA, serial SCSI

Disk Scheduling

- Disks poor at random R/W, better at sequential
- Seeking activity important factor in performance
  - Minimize disk seek time (moving from track to track)
- Minimize rotational latency (waiting for disk to rotate the desired sector under read/write head)
- Example: Openoffice startup long!
  - Excessive seeks as loader fixes relocations
  - Shared objs (many!) mapped and fixing relocations causes page faults: many seeks

Some Disk Scheduling Algs.

- FCFS
- Shortest Seek Time First (SSTF)
- Elevator or SCAN: Disk arm starts at one end of disk and moves towards other end, servicing requests as it goes
  - Reverses direction at end of disk
- C-SCAN: same as SCAN, except head returns to cylinder 0 at end of the disk
- C-LOOK: same as C-SCAN, except head only travels as far as the last request in each direction

Linux Disk Scheduling

- (Linus) Elevator (default till '03)
- Deadline
  - Imposes a deadline on all I/O operations to prevent resource starvation.
- Anticipatory (default '04 - '06; now removed)
  - pauses for a short time (a few ms) after a read operation in anticipation of other close-by read reqs
- Completely Fair Queuing (CFQ) (default from '06)
  - allocates timeslices for each of the per-process queues (synch/asynch) for access to the disk
- Null

Test 1. Writes-Starving-Reads

- In background, perform a streaming write, such as:

```
while true
do
        dd if=/dev/zero of=file bs=1M
done
```

- Meanwhile, time how long a simple read of a 200MB file takes:

```
time cat 200mb-file > /dev/null
```

(from a Linux kernel mailing list discussion)

Test 2. Effects of High Read Latency

- Start a streaming read in the background:

```
while true
do
        cat big-file > /dev/null
done
```

- Meanwhile, measure how long it takes for a read of every file in the kernel source tree to complete:

```
time find . -type f -exec cat '{}' ';' > /dev/null
```

(from a Linux kernel mailing list discussion)

Performance Results

I/O Scheduler and Kernel	**Test 1**	**Test 2**
Linus Elevator on 2.4	45.0 secs	30 mins, 28 secs
Deadline I/O Scheduler on 2.6	40.0 secs	3 mins, 30 secs
Anticipatory I/O Scheduler on 2.6	4.6 secs	15 secs

（from a Linux kernel mailing list discussion）

Summary

- We looked at the basic API for storage
- We discussed layering
- We started looking at the physical layer (disk)