Intro to Distributed Systems, Distributed File Systems

ECE595
April 17

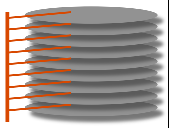
Y. Charlie Hu

Online Course Survey Opens

- Open till Apr 30

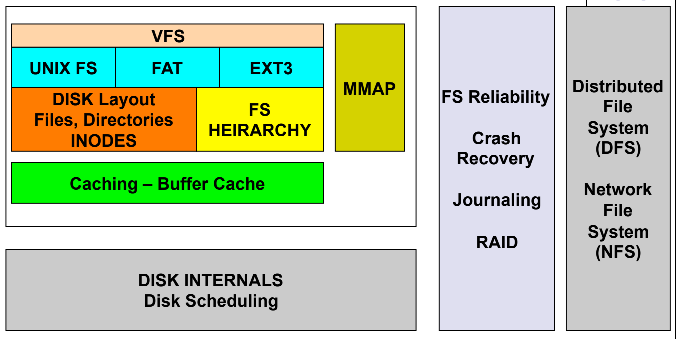
[week9] Disk mechanics & performance

- Platter / Head / Tracks / Sectors / Cylinders
- Rotation 1000's of RPM (7200, 10k, 15k)
- Avg seek 5-10 ms



- Assume
  - 255 heads *38913 tracks * 63 sectors * 512 bytes = 320GB
  - Seek time =6ms, 7200 RPM ➔ rotational latency = 8ms
- *Block access time = seek time + rotational latency + reading time*
  - Accessing a random block: 6ms + 4 ms + 8ms/63 = 638ms/63
  - Accessing the block after: 8ms/63
- Implications?

FS topics we have covered


VFS
UNIX FS
FAT
EXT3
DISK Layout Files, Directories INODES
FS HEIRARCHY
MMAP
Caching – Buffer Cache
DISK INTERNALS Disk Scheduling
FS Reliability
Crash Recovery
Journaling
RAID
Distributed File System (DFS)
Network File System (NFS)

Misc

- Rumor: “~~final will be harder than midterm~~”

  Lie!

- Today’s topic:
  - Advanced
  - But not hard: Demonstrate what great things we can build by putting concepts/techniques we learned in ece595 together!

Distributed Systems (ECE 673)

- Collection of computer nodes
- Connected by a network
- Running software that makes it appear as a single system
  - Nodes function together to achieve a common goal
  - Nodes communicate and coordinate their actions by passing messages

Examples

- WWW
- Web 2.0 / Online services
  - Search, Social networks, Blogs, Wikis, Video Sharing
- Air-traffic control
- Stock brokerage systems
- Banking
- Distributed supercomputing (Grid computing)
- *Distributed file systems*
- DNS
- Peer-to-peer systems (Gnutella, BitTorrent, …)

Distributed systems

- Key differences from centralized systems
  - No shared memory
  - Communication: delays, unreliable
  - No common clock
  - Independent node failure modes
  - Hardware/software heterogeneity

Distributed systems (cont)

- Need to revisit many OS issues
  - Distributed synchronization
  - Distributed deadlock detection
  - Distributed memory management
  - Distributed file systems
- Plus a number of new issues
  - Distributed agreement (e.g. leader election)
  - Distributed event ordering (e.g. newsgroup)
  - (Partial) Failure recovery

Pillars of Distributed Systems

- Operating systems (ECE 595)
- Networking (ECE 595)

Network protocols

- Protocols are the key to networks
- What is a *protocol*?
  - An agreement between the parties on the network about how information will be transmitted between them
- Examples
  - e.g. HTTP, mail (SMTP), file transfer, remote login
- *Protocols are built up in layers (why?)*
  - ISO OSI model
  - TCP/IP

TCP connection establishment

- "Coordinated-attack problem"

"several generals are planning to attack (from different directions) against a common enemy. It is known the only way for the attack to succeed, is if all the generals attack. If only some attack, they will be destroyed, for many reasons… The generals are located in different places, and can communicate only via messengers. But messengers can be caught…"

Pillars of Distributed Systems

- Operating systems (ECE 595)
- Networking (ECE 595)

- Distributed systems
  - Needs a basic model for parties to talk to each other

Client-Server Model

- Client-server model is the dominating model for constructing distributed applications and services during the Internet explosion era (1994-2000)

- It still is today
  - E.g. cloud computing

Why Client-Server Model?

- Networking protocols are responsible for data transmission
  - No mechanism for automatically starting processes
  - No policies regarding who starts and who waits
  - ➔ There is a need for a task interaction model

- Goal: enable 2 processes to *sync.* in order to perform a cooperative computation
  - One of the processes – server - must wait for the other
  - The other - client - initiates interaction and server replies

Client-Server Model (cont)

- Requires 2 mandatory types of messages
  - request (client to server)
  - reply (server to client)
- Requires 2 types of systems calls
  - send (dst, buf)
  - recv (src, buf)
- Basic operations

client	server
send(req)	.
.	.
.	recv(req)
wait	processing req
.	send (rep)
recv(rep)	.

Client-Server Comm. Semantics

- Addressing
  - machine_ID:process_ID (location-dependent)
  - machine_ID:port (e.g. sockets, location-dependent, process-independent)
  - logical server name – location independent, but requires name server
- What about the server side?
  - on *recv*, source (client) address may be ANY
  - How does server find out client address?

Client-Server Comm. Semantics (cont)

- Blocking (synchronous) vs. non-blocking send
  - *fully blocking send*: wait until message arrived
  - *blocking send* (OS view): wait until message sent
  - *blocking send (app view)*: copy to kernel space, wait only until copy completed (motivation?)
  - *nonblocking send*: return immediately, interrupt when sent (implication?)
  - Unix socket send
    - Default: blocking till copied (what if not enough buffer space?)
    - Nonblocking mode (via fcntl): return with EWOULDBLOCK if no space, use poll or select

Client-Server Comm. Semantics (cont)

- Blocking (synchronous) vs. non-blocking recv
  - *blocking recv:* wait until message arrived (common)
  - *nonblocking recv*: return immediately
  - Unix socket supports both modes

More Client-Server Issues

- How to handle concurrency efficiently for client-server applications (e.g., on server)?
  - multiple processes
    - Blocking recv, blocking send
  - multithreading often most convenient
    - Blocking recv, blocking send
  - for highest performance $\rightarrow$ I/O mutiplexing
    - select/poll in Unix socket, kevent in FreeBSD

Distributed File Systems

- You can login to any instructional machine on campus, your home dir is always there!
- but sometimes it is very slow…

DFS

- Definition: a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources
- Many DFS have been proposed, developed
- Only one is widely used

DFS uses client/server model

- What is the service?
  - File service
- What is the client interface?
  - a set of primitive *file operations* (create, delete, read, write)
- Client interface of a DFS should be *transparent*, i.e., not distinguish between local and remote files

What is NFS (Network File System)?

- First commercially successful network file system:
  - Developed in 1984 by Sun Microsystems for their diskless workstations
  - Designed for robustness and "adequate performance"
  - Sun published all protocol specifications
  - Many many implementations
  - Widely used today

Demo

NFS Design Objectives

- ***Machine and Operating System Independence***
  - Could be implemented on low-end machines of the mid-80's
- ***Transparent Access***
  - Remote files should be accessed in exactly the same way as local files
- ***Fast Crash Recovery***
  - Major reason behind stateless design
- ***"Reasonable" performance***
  - Robustness and preservation of UNIX semantics were much more important

Two naming schemes

- Files named by combination of their host name and local name; guarantees a unique systemwide name
  - Neither location transparent
  - Nor location independent
- "Attach" remote directories to local directories, giving the appearance of a coherent directory tree, e.g. Sun's NFS
  - Early NFS: only previously mounted remote directories can be accessed transparently
  - automount

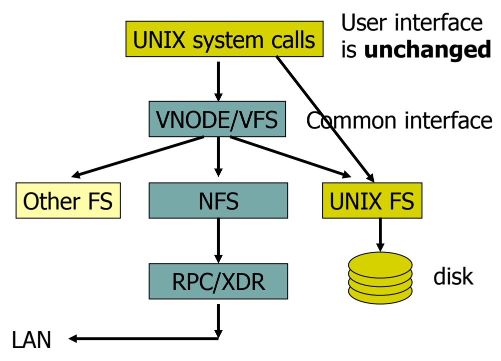
Implementation of transparency

- "*All computer science problems can be solved with an extra level of indirection*"

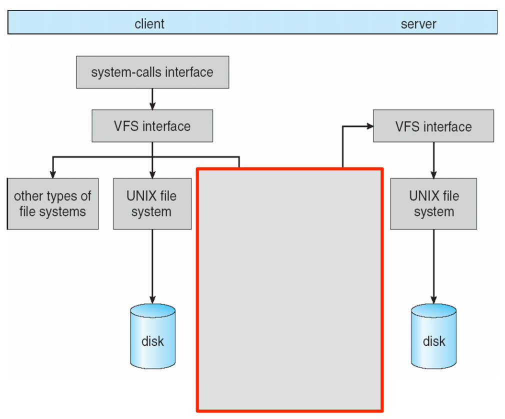
  -- David Wheeler

- What were the earlier manifestations of this in this class?

Client Side



Schematic View of NFS Architecture



How to identify files in NFS?

- Can we still use inode?
- NFS use File handles
- **File handle** consists of
  - ***Filesystem id*** identifying disk partition
  - ***i-node number*** identifying file within partition
  - ***i-node generation number*** changed every time i-node is reused to store a new file

Filesystem id	i-node number	i-node generation number

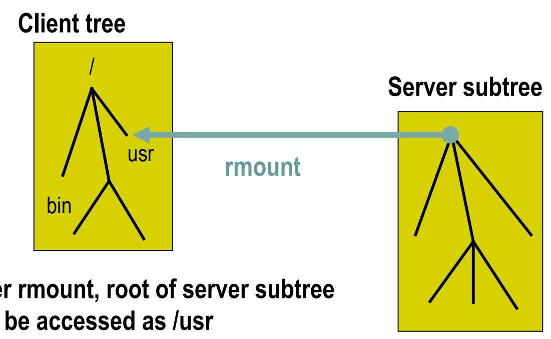
[week10] UFS Mounting

- In UNIX, file systems can be mounted at any dir
- Implementation
  - Setting a flag in the in-memory copy of the inode
  - A field points to an entry in the mount table
  - Mount table contains a pointer to the superblock of mounted file system

Client Side (II)

- Transparent interface to NFS achieved by mapping between remote file names and remote file addresses via ***remote mount***
  - Extension of UNIX mount
  - Specified in a ***mount table***
  - Makes a remote subtree appear as part of a local subtree

Client side (III): Remote Mount



After rmount, root of server subtree can be accessed as /usr

NFS Design Objectives

- ***Machine and Operating System Independence***
  - Could be implemented on low-end machines of the mid-80's
- ***Transparent Access***
  - Remote files should be accessed in exactly the same way as local files
- ***Fast Crash Recovery***
  - Major reason behind stateless design
- ***"Reasonable" performance***
  - Robustness and preservation of UNIX semantics were much more important

To be cont.

How to identify files in NFS?

- Can we still use inode?
- NFS use File handles
- **File handle** consists of
  - ***Filesystem id*** identifying disk partition
  - ***i-node number*** identifying file within partition
  - ***i-node generation number*** changed every time i-node is reused to store a new file

Filesystem id	i-node number	i-node generation number

Stateful vs. stateless protocols

- Stateful:
  - fd = open("/aaa/bbb/ccc", …);
  - read(fd, …)
- ***Stateless:***
  - fhandle = open("/aaa/bbb/ccc", …);
  - read(fhandle, …)
  - Each procedure call contains ***all the information necessary to complete the call***
  - Server maintains no "between call" information

Advantages of stateless

- Crash recovery is very easy:
  - When a server crashes, client just resends request until it gets an answer from the rebooted server
  - Client cannot tell difference between a server that has crashed and recovered and a slow server
- Simplifies the protocol
  - Client can always repeat any request

Consequences of stateless

- read and writes calls must specify offset
  - Server does not keep track of current position in the file
- But user will still use conventional UNIX APIs
- How should UNIX APIs be translated?
  - open() / close()
  - read() / write()

Remote lookup (I)

- Returns a **file handle** instead of a file desc.
  - File handle specifies *unique location* of file
- **lookup(dirfh, name)** *returns* **(fh, attr)**
  - Returns file handle **fh** and attributes of named file in directory **dirfh**
  - Fails if client has no right to access directory **dirfh**

Remote lookup (II)

- In Virtual File System, a single open call such as

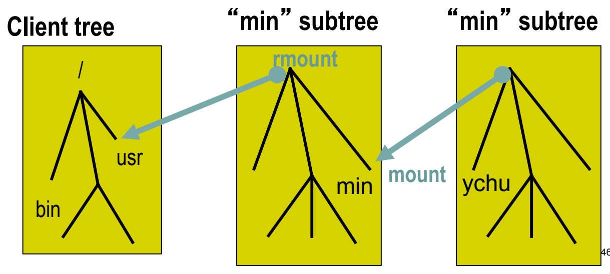
  open("/usr/joe/6360/list.txt", ….)

  will be result in several calls to lookup

  lookup(rootfh, "usr") returns (fh0, attr)
  lookup(fh0, "joe") returns (fh1, attr)
  lookup(fh1, "6360") returns (fh2, attr)
  lookup(fh2, "list.txt") returns (fh, attr)

The lookup call (III)

- Why all these steps?
  - Any of components of **/usr/min/ychu/list.txt** could be a ***mount point***



NFS Design Objectives

- ***Machine and Operating System Independence***
  - Could be implemented on low-end machines of the mid-80's
- ***Transparent Access***
  - Remote files should be accessed in exactly the same way as local files
- ***Fast Crash Recovery***
  - Major reason behind stateless design
- ***"Reasonable" performance***
  - Robustness and preservation of UNIX semantics were much more important

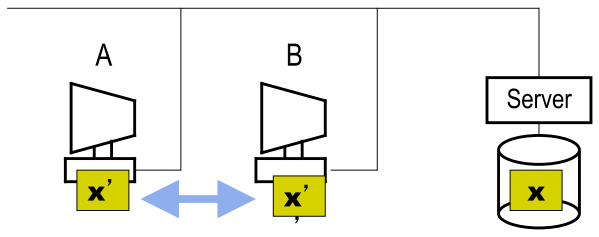
NFS Performance Optimization

- *Caching*:
  - Reduce network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled locally
    - Where else have we talked about caching in ece469?
- How is caching in NFS different from in FS?

Basic Caching scheme in DFS

- Files identified with one master copy residing at the server machine, but copies of (parts of) the file are scattered in different caches
- Cache-Consistency Problem

Cache Consistency Problem



Inconsistent updates

Cache Update Policy

- **Write-through** – write data through to master’s copy as soon as they are placed on any cache
  - Reliable
  - but poor performance
    - Only reads benefit

Cache Update Policy

- **Delayed-write** – modifications written to the cache and written through to the server later
  - Write accesses complete quickly
    - data may be overwritten before written back, and so aggregated
  - Poor reliability
    - unwritten data will be lost whenever client crashes
  - Variation (used in NFS)
    - scan cache at regular intervals and flush dirty blocks since the last scan
    - *write-on-close*, writes data back to the server when the file is closed.

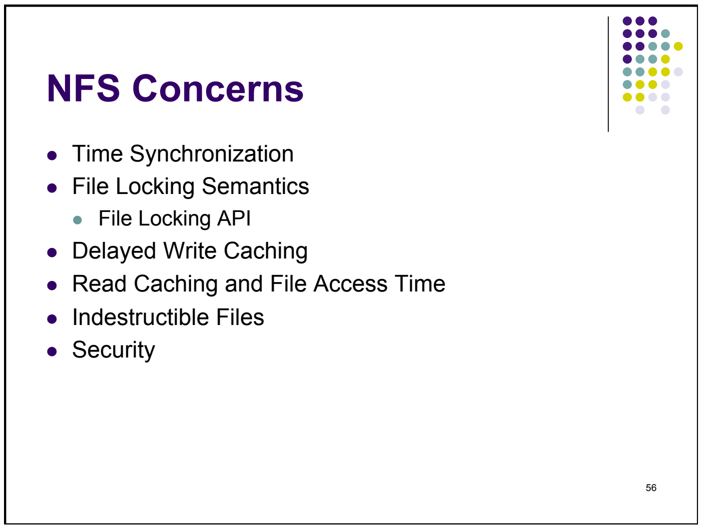
Summary: NFS Design Objectives

- ***Machine and Operating System Independence***
  - Could be implemented on low-end machines of the mid-80's
- ***Transparent Access***
  - Remote files should be accessed in exactly the same way as local files
- ***Fast Crash Recovery***
  - Major reason behind stateless design
- ***"Reasonable" performance***
  - Robustness and preservation of UNIX semantics were much more important

Questions?

- Online Course evaluation
  - Open till Apr 30

Questions?



NFS Concerns

- Time Synchronization
- File Locking Semantics
  - File Locking API
- Delayed Write Caching
- Read Caching and File Access Time
- Indestructible Files
- Security