CSE 486/586 Distributed Systems
Distributed File Systems

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Local File Systems

- File systems provides file management.
  - Name space
  - API for file operations (create, delete, open, close, read, write, append, truncate, etc.)
  - Physical storage management & allocation (e.g., block storage)
  - Security and protection (access control)
- Name space is usually hierarchical.
  - Files and directories
- File systems are mounted.
  - Different file systems can be in the same name space.

Traditional Distributed File Systems

- Goal: emulate local file system behaviors
  - Files not replicated
  - No hard performance guarantee
- But,
  - Files located remotely on servers
  - Multiple clients access the servers
- Why?
  - Users with multiple machines
  - Data sharing for multiple users
  - Consolidated data management (e.g., in an enterprise)

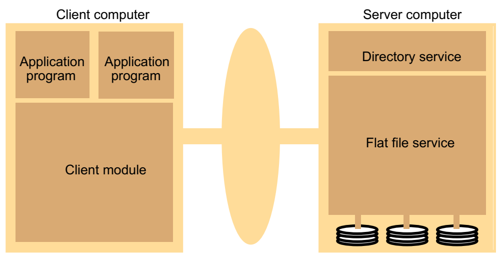
Requirements

- Transparency: a distributed file system should appear as if it were a local file system
  - Access transparency: it should support the same set of operations, i.e., a program that works for a local file system should work for a DFS.
  - (File) Location transparency: all clients should see the same name space.
  - Migration transparency: if files move to another server, it shouldn't be visible to users.
  - Performance transparency: it should provide reasonably consistent performance.
  - Scaling transparency: it should be able to scale incrementally by adding more servers.

Requirements

- Concurrent updates should be supported.
- Fault tolerance: servers may crash, msgs can be lost, etc.
- Consistency needs to be maintained.
- Security: access-control for files & authentication of users

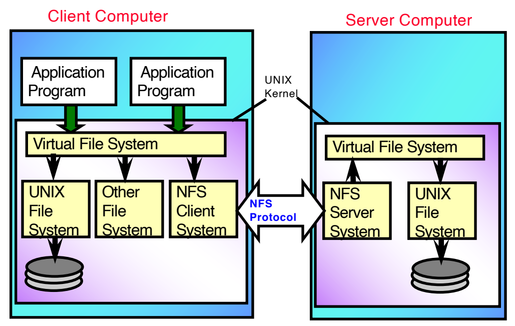
File Server Architecture



Components

- Directory service
  - Meta data management
  - Creates and updates directories (hierarchical file structures)
  - Provides mappings between user names of files and the unique file ids in the flat file structure.
- Flat file service
  - Actual data management
  - File operations (create, delete, read, write, access control, etc.)
- These can be independently distributed.
  - E.g., centralized directory service & distributed flat file service

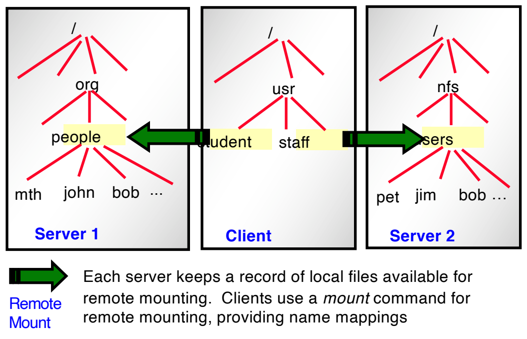
Sun NFS



VFS

- A translation layer that makes file systems pluggable & co-exist
  - E.g., NFS, EXT2, EXT3, ZFS, etc.
- Keeps track of file systems that are available locally and remotely.
- Passes requests to appropriate local or remote file systems
- Distinguishes between local and remote files.

NFS Mount Service



Each server keeps a record of local files available for remote mounting. Clients use a *mount* command for remote mounting, providing name mappings

NFS Basic Operations

- Client
  - Transfers blocks of files to and from server via RPC
- Server
  - Provides a conventional RPC interface at a well-known port on each host
  - Stores files and directories
- Problems?
  - Performance
  - Failures

Improving Performance

- Let's cache!
- Server-side
  - Typically done by OS & disks anyway
  - A disk usually has a cache built-in.
  - OS caches file pages, directories, and file attributes that have been read from the disk in a *main memory buffer cache*.
- Client-side
  - On accessing data, cache it locally.
- What's a typical problem with caching?
  - Consistency: cached data can become stale.

(General) Caching Strategies

- Read-ahead (prefetch)
  - Read strategy
  - Anticipates read accesses and fetches the pages following those that have most recently been read.
- Delayed-write
  - Write strategy
  - New writes stored locally.
  - Periodically or when another client accesses, send back the updates to the server
- Write-through
  - Write strategy
  - Writes go all the way to the server's disk
- This is not an exhaustive list!

NFS V3 Client-Side Caching

- We'll mainly look at NFS V3.
- Write-through, but only at close()
  - Not every single write
  - Helps performance (reduces network activities & traffic)
- Multiple writers
  - No guarantee
  - Could be any combination of (over-)writes
- Leads to inconsistency

Validation

- A client periodically checks with the server about cached blocks.
- Each block has a timestamp.
  - If the remote block is new, then the client invalidates the local cached block.
- Always invalidate after some period of time
  - 3 seconds for files
  - 30 seconds for directories
- Written blocks are marked as "dirty."

Failures

- Two design choices: stateful & stateless
- Stateful
  - The server maintains all client information (which file, which block of the file, the offset within the block, file lock, etc.)
  - Good for the client-side process (just send requests!)
  - Becomes almost like a local file system (e.g., locking is easy to implement)
- Problem?
  - Server crash → lose the client state
  - Becomes complicated to deal with failures

Failures

- Stateless
  - Clients maintain their own information (which file, which block of the file, the offset within the block, etc.)
  - The server does not know anything about what a client does.
  - Each request contains complete information (file name, offset, etc.)
  - Easier to deal with server crashes (nothing to lose!)
- NFS V3's choice
- Problem?
  - Locking becomes difficult.

NFS V3

- Client-side caching for improved performance
- Write-through at close()
  - Consistency issue
- Stateless server
  - Easier to deal with failures
  - Locking is not supported (later versions of NFS support locking though)
- Simple design
  - Led to simple implementation, acceptable performance, easier maintenance, etc.
  - Ultimately led to its popularity

NFS V4

- Stateful system
  - New APIs: open() and close()
  - Locking is supported through lock(), lockt(), locku(), renew()
  - Supports read/write locks, call backs etc.
- Effective use of client side caching
- Version 4.1 (pNFS)
  - Parallel NFS supports parallel file I/O
  - File is striped and stored across multiple servers
  - Metadata and data are separated

CSE 486/586 Administrivia

- Mid-semester grades will be posted by Friday.
- PA3 is due on Friday.
- Academic integrity

Brief Intro to Data Centers

- The rest of the semester deals with data centers a lot.

Data Centers

Data Centers

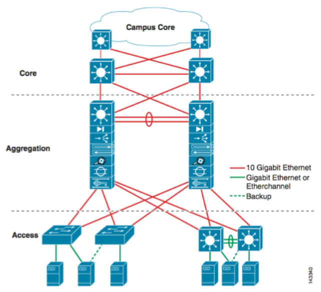
- Hundreds of Locations in the US

Inside

- Servers in racks
  - Usually ~40 blades per rack
  - ToR (Top-of-Rack) switch
- Incredible amounts of engineering efforts
  - Power, cooling, etc.

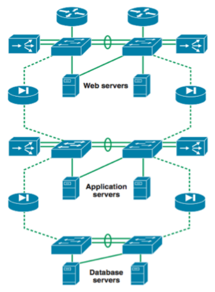
Inside

- Network



Inside

- 3-tier for Web services

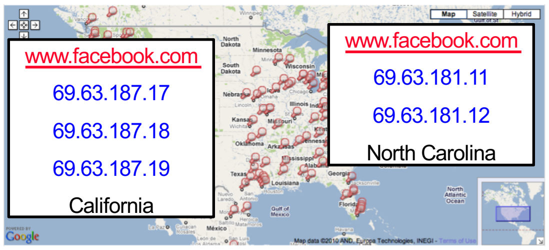


Web Services

- Amazon, Facebook, Google, Twitter, etc.
- World-wide distribution of data centers
  - Load balance, fault tolerance, performance, etc.
- Replicated service & data
  - Each data center might be a complete stand-alone web service. (It depends though.)
- At the bare minimum, you're doing read/write.
- What needs to be done when you issue a read req?
  - Server selection
- What needs to be done when you issue a write req?
  - Server selection
  - Replicated data store management

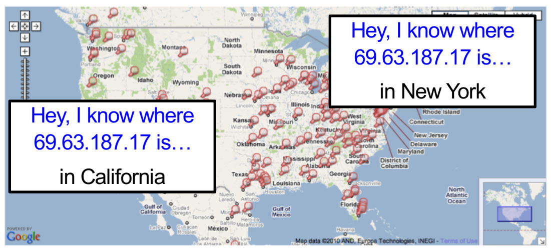
Server Selection Primer

- Can happen at multiple places
- Server resolution process: DNS -> External IP -> Internal IP
- DNS



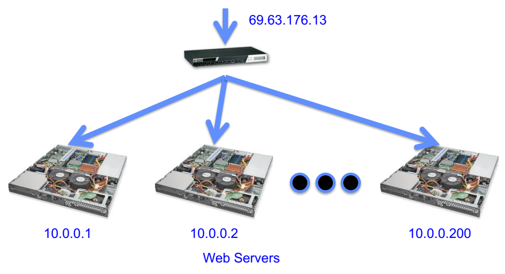
IP Anycast

- BGP (Border Gateway Protocol) level

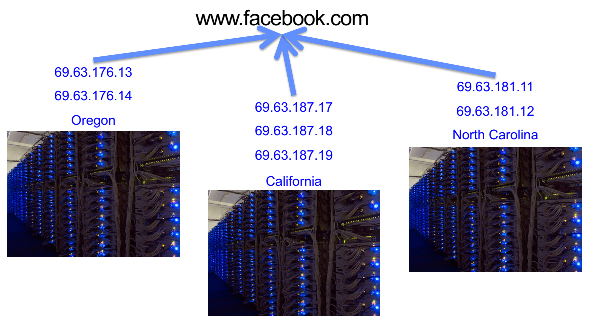

Hey, I know where 69.63.187.17 is…
in New York
Hey, I know where 69.63.187.17 is…
in California

Inside

- Load balancers



Example: Facebook



Core Issue: Handling Replication

- Replication is (almost) inevitable.
  - Failures, performance, load balance, etc.
- We will look at this in the next few weeks.
- Data replication
  - Read/write can go to any server.
  - How to provide a consistent view? (i.e., what consistency guarantee?) linearizability, sequential consistency, causal consistency, etc.
  - What happens when things go wrong?
- State machine replication
  - How to agree on the instructions to execute?
  - How to handle failures and malicious servers?

Summary

- NSF
  - Caching with write-through policy at close()
  - Stateless server till V3
  - Stateful from V4
  - 4.1 supports parallel I/O

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).