Today's Objectives

- Distributed File Systems
- Timing

Sakai Poll

- Which class would you prefer to use for the exam?
  - Wednesday
  - Friday
- Answer by 5:30 p.m.

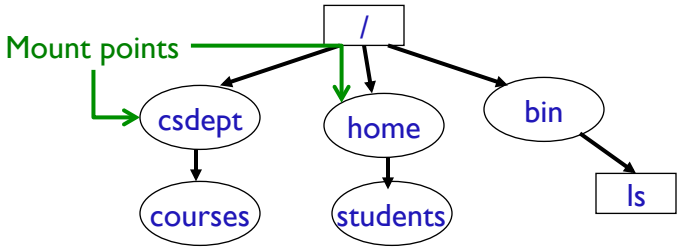
Extra Office Hours

- Today: ~2:45 – 4:30 p.m.
  - CSCI111 students taking exam at 2:30
    - Get priority

Review

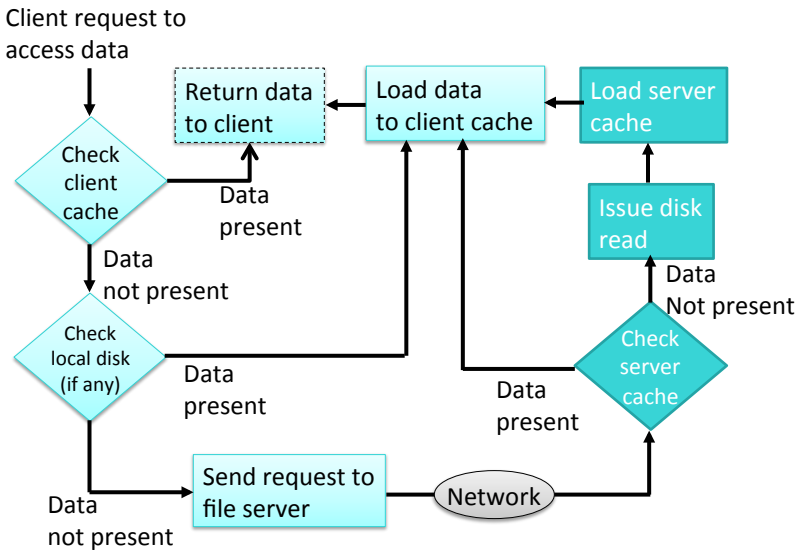
- What is the motivation for a distributed file system (DFS)?
- What does it mean for a file system to be distributed?
- How does a DFS make remote files look the same as local files?

Distributed File System Structure



- Perform ***mount*** operation to attach remote file system into local namespace
  - E.g., `/home/students` is actually a file on remote machine
  - Maps to `hydros.cs.wlu.edu:/exports/home/students`
- ***Mounting*** helps to combine files/directories in different systems and form a single file system structure

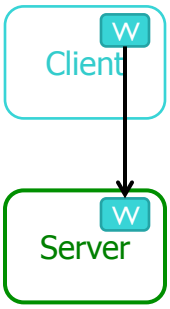
DFS Data Access



Writing Policy

When should a modified cache content be transferred to the server?
What are the tradeoffs?

Client made changes to a file



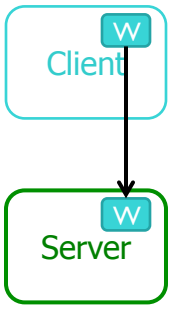
Writing Policy

When should a modified cache content be transferred to the server?

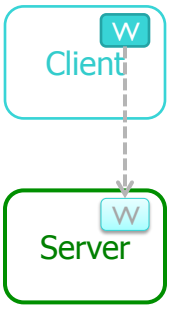
- Write-through policy
  - Immediate writing at server when cache content is modified.
  - Advantage: reliability, crash of cache (client) does not mean loss of data.
  - Disadvantage: Several writes for each small change.
- Write-back policy
  - Write at the server, after a delay.
  - Advantage: small/frequent changes do not increase network traffic.
  - Disadvantage: less reliable, susceptible to client crashes.
- Write at the time of file closing
  - Advantage: even less network traffic
  - Disadvantage: even less reliable, more susceptible to client crashes.

Write-Back vs Write-Through Caching

Write-Through

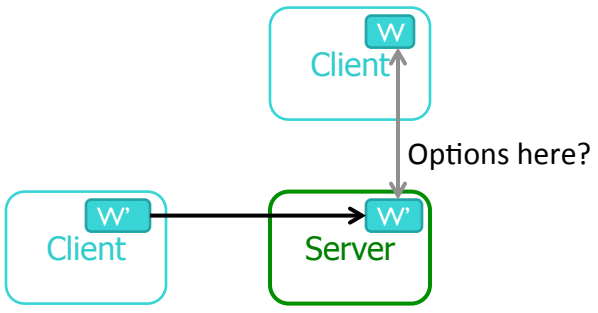


Write-Back



Cache Consistency

When should a modified source content be transferred to the cache?
What are the tradeoffs?



Cache Consistency

When should a modified source content be transferred to the cache?

- Server-initiated policy:
  - Server cache manager informs client cache managers that can then retrieve the data.
- Client-initiated policy:
  - Client cache manager checks the freshness of data before delivering to users. Overhead for every data access.
- Concurrent-write sharing policy:
  - Multiple clients open the file, at least one client is writing.
  - File server asks other clients to purge/remove the cached data for the file, to maintain consistency.
- …

Cache Consistency

When should a modified source content be transferred to the cache?

- Sequential-write sharing policy: a client opens a file that was recently closed after writing.
  - This client may have outdated cache blocks of the file
    - Other client might have modified the file contents
    - Use time stamps for both cache and files
    - Compare the time stamps to know the freshness of blocks.
  - The other client (which was writing previously) may still have modified data in its cache that has not yet been updated on server due to delayed writing.
    - Server can force the previous client to flush its cache whenever a new client opens the file.

Availability

- **Intention**: overcome failure of servers or network links
- Solutions?
- Tradeoffs?

Availability

- **Intention**: overcome failure of servers or network links
- Solution: replication, i.e., maintain copies of files at different servers.
- Issues:
  - Maintaining consistency
  - Detecting inconsistencies, if they happen despite best efforts. Possible reasons for such inconsistencies:
    - Replica is not updated due to a server failure or a broken network link.
- Inconsistency problems and their recovery may reduce the benefit of replication.

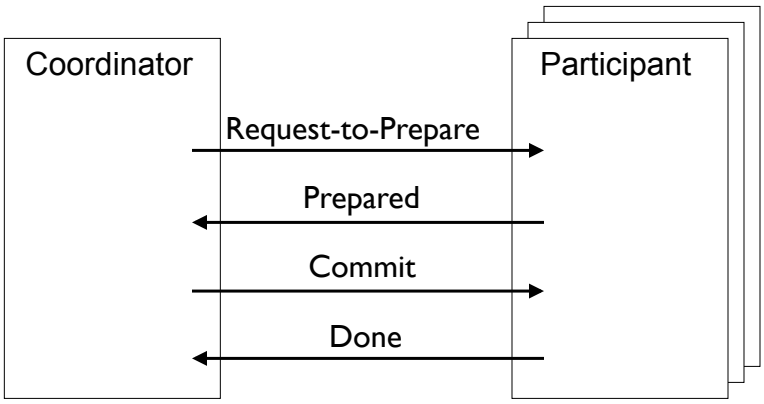
Availability: Replication Alternatives

- Replication unit: a file
  - Replicas of a file in a directory may be handled by different servers, requiring extra name resolutions to locate the replicas.
- Replication unit: group of files
  - Advantage: process of name resolution, etc., to locate replicas can be done for a set of files and not for individual files.
  - Disadvantage: wasteful of disk space if only very few of this group of files is needed by users often.

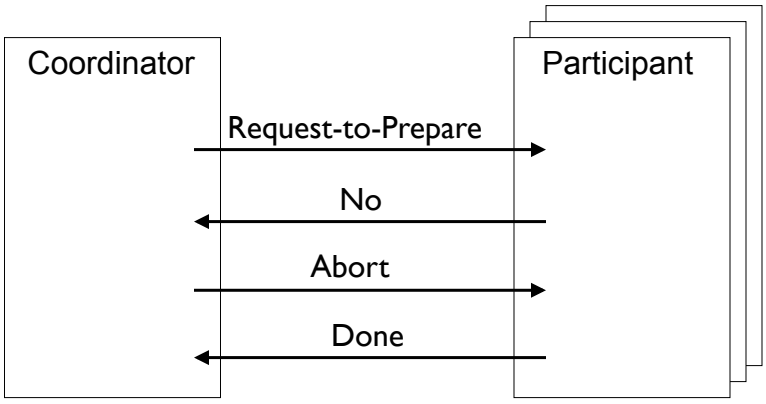
Replica Management: Two-Phase Commit

- Standard protocol for making commit and abort atomic
- Use a persistent, stable log on each machine to keep track of whether commit has happened
  - If a machine crashes, when it wakes up, it checks its log to recover state of world at time of crash
- Prepare Phase:
  - Global coordinator requests that all participants will promise to commit or rollback the transaction
  - Participants record promise in log, then acknowledge
  - If anyone votes to abort, coordinator writes "Abort" in its log and tells everyone to abort; each records "Abort" in log
- Commit Phase:
  - After all participants respond that they are prepared, then the coordinator writes "Commit" to its log
    - Then asks all nodes to commit; they respond with ack
    - After receive acks, coordinator writes "Got Commit" to log

Case 1: Commit



Case 2: Abort



Replica Management: Other Schemes

- Weighted votes:
  - A certain number of votes r or w is to be obtained before reading or writing.
- Current synchronization site (CSS):
  - Designate a process/site to control the modifications.
  - File open/close are done through CSS.
  - CSS can become a bottleneck.

Scalability

- Goal: Ease of adding more servers and clients with respect to the problems/design issues discussed before, such as caching, replication management, etc.

Scalability

- Goal: Ease of adding more servers and clients with respect to the problems / design issues discussed before such as caching, replication management, etc.
- Server-initiated cache invalidation scales up better
- Using the client's cache:
  - A server serves only X clients.
  - New clients (after the first X) are informed of the X clients from whom they can get the data (sort of chaining/hierarchy).
  - Cache misses & invalidations are propagated up and down this hierarchy, i.e., each node serves as a mini-file server for its children.
- Structure of a server:
  - I/O operations through threads (light weight processes) can help in handling more clients.

Building a Distributed File System

- Debate in late 1980's, early 1990's:
  - Stateless vs. stateful file server
- Sun NFS: stateless server
  - Only store contents of files + soft state (for performance)
  - Crash recovery simple operation
  - All RPC's ***idempotent*** (no state)
    - "At least once" RPC semantics sufficient
  - Server unaware of users accessing files
- Clients have to check with server periodically for the uncommon case
  - When directory/file has been modified

Sun NFS

- Sun Microsystem's Network File System
  - Widely adopted in industry and academia since 1985
  - (we use it)
- All NFS implementations support NFS protocol
  - Currently on version 4
  - Protocol is a set of **RPCs** that provide mechanisms for clients to perform operations on remote files
  - OS-independent but originally designed for UNIX

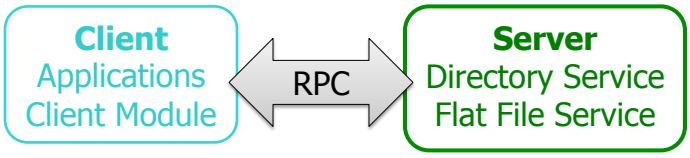
File Service Architecture

- Separate main concerns in providing access to files by structuring file service as three components:

Flat file service	Implement operations on contents of files; uses Unique File Identifiers (UFID)
Directory service	Provides mapping between *text names* for files and their *UFIDs*; Used by clients to create, modify, manipulate directories
Client module	Runs in each computer, integrates and extends flat file service (using RPC) and directory service operations using API that user-level programs can use

- NFS roughly follows this model

NFS Architecture



- Client-server design
- Server module resides in kernel on each NFS server
- Client modules translate requests for remote files and are passed to server module at computer holding the relevant file system
- Clients and servers communicate using Sun's RPC system

NFS Protocol

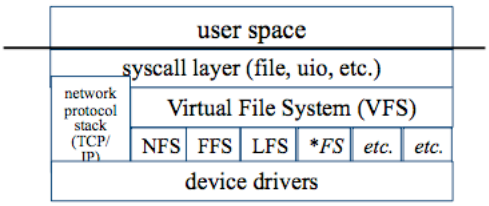
- Network protocol
- Layered above TCP/IP
  - NSF 4: requires TCP as a transport
  - Original implementations (2 & 3) use UDP datagram transport for low overhead
    - Maximum IP datagram size was increased to match FS block size, to allow send/receive of entire file blocks
- A set of message formats and types
  - Client issues a request message for a service operation
  - Server performs requested operation and returns a reply message with status and (perhaps) requested data

NFS protocol architecture

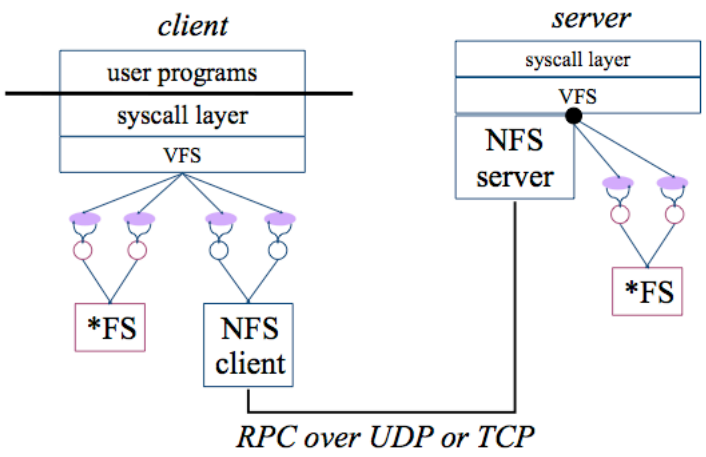
- I/O RPCs are ***idempotent***
  - multiple repetitions have same effect as one
  - lookup(handle, "emacs") generally returns same result
  - read(file-handle, offset, length) ⇒ bytes
  - write(file-handle, offset, buffer, bytes)
- RPCs do not create server-memory state
  - no RPC calls for open()/close()
  - write() succeeds (to disk) or fails before RPC completes

VFS: The File System Switch

- In 1985 Sun introduced ***virtual file system*** interface to accommodate diverse file system types cleanly
  - Allows diverse file systems to coexist
- No effect on the system call interface



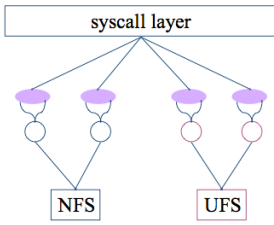
Network File System (NFS)



VFS=Virtual File System

VFS: Vnodes

- Every file or directory in active use is represented by a virtual node or ***vnode*** object in memory
  - Each file system maintains a cache of its vnodes
  - Each vnode has a standard file attribute struct
  - Each standard struct points at file-system-specific file attribute struct



Standard Struct
FS-specific Struct

NFS file handles

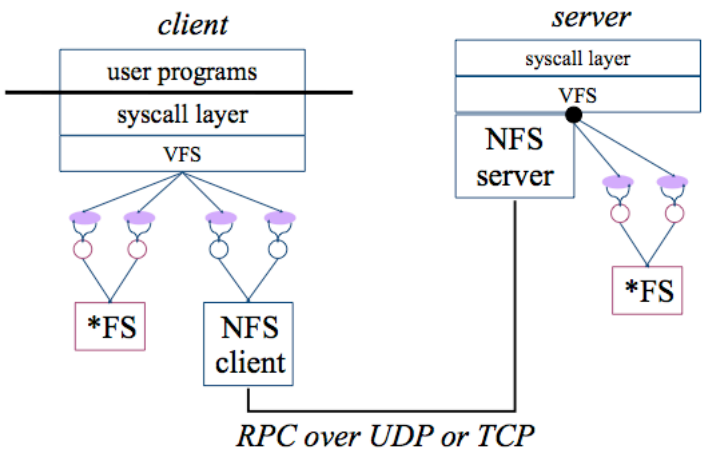
- Goals
  - Reasonable size
  - Quickly map to file on server
  - "Capability"
    - Hard to forge, so possession serves as "proof"
- Implementation (inode #, inode generation #)
  - inode # - small, fast for server to map onto data
  - "inode generation #" - must match value stored in inode
    - "unguessably random" number chosen in create()

Pathname Traversal

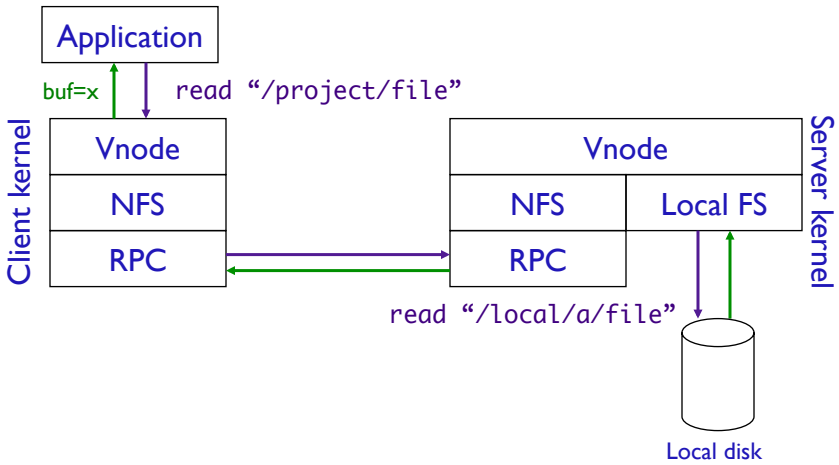
- When a pathname is passed as an argument to a system call, syscall layer "converts" it to a vnode
- Pathname traversal is a sequence of lookup calls to descend the file tree to the named file
- Issues:
  - Crossing mount points
  - Finding root vnode
  - Locking
  - Caching name->vnode translations

```
open("/tmp/zot")
vp = get vnode for / (rootdir)
vp->vop_lookup(&cvp, "tmp");
vp = cvp;
vp->vop_lookup(&cvp, "zot");
```

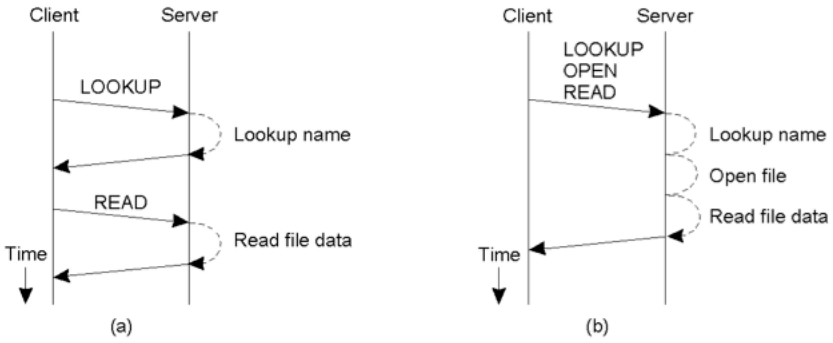
Network File System (NFS)



NFS Data Access Model



Two Options for NFS Lookup/Read



NFSv3 NFSv4

Stateless NFS

- NFS server maintains no in-memory hard state
  - Only hard state is stable file system image on disk
  - No record of clients or open files
  - No implicit arguments to requests (no server-maintained file offsets)
  - No write-back caching on server
  - No record of recently processed requests
- Why? Simple recovery!

Recovery in NFS

- If server fails and restarts, no need to rebuild in-state memory state on server
  - Client reestablishes contact
  - Client retransmits pending requests
- Classical NFS uses UDP
  - Server failure is transparent to client since there is no "connection"
  - Sun RPC masks network errors by retransmitting requests after an adaptive timeout
    - Dropped packets are indistinguishable from crashed server to client

NFS Server Caching

- Cache read results, writes, directory operations
- Write-through cache vs. write-back cache?
  - **Write through**: Each update written to disk immediately
  - When write operation returns, client is guaranteed stable update
- Pros:
  - Stateless (easy to implement), no data lost on crash
- Cons:
  - Slow: client must wait for disk write

Drawbacks

- Stateless nature has obvious advantages but also some drawbacks
  - Recovery by retransmission constrains server interface
    - "Execute mostly once" semantics = send and pray
    - Executions usually only happen once, but not guaranteed
  - Update operations are disk-limited (write-through cache)
  - Server cannot help in client cache consistency

NFS Client Caching

- Clients cache read, writes, and directory ops
  - What if multiple people updating the same file at the same time? Consistency problems!
- NFS approach:
  - Server maintains last modification time/per file
  - Client remembers time it initially retrieved data
  - On file access, client checks timestamp against server (every 3-30 seconds)
    - Unnecessary timestamp checking
    - How long to set the timeout? What is the tradeoff?

NFS Replication

- As originally specified, NFS did not support data replication
- More recent versions of NFS support replication via a mechanism called Automounter
  - Allows remote mount points to be specified using a set of servers
  - Manually propagate modifications to replicas
- Intended primarily for READ-ONLY files

NFS Security

- NFS uses underlying Unix file protection on servers for access checks
- In early NFS, mutual trust assumed among all participating machines
  - User identity determined by client machine and accepted without further server validation
- **Kerberos**: computer network authentication protocol
  - Allows nodes communicating over non-secure network to prove their identity to one another securely
  - Port 88
- File data in RPC packets is not encrypted➔NFS is still vulnerable

Access Control

- Users have various rights w.r.t. a file
  - Read, write, update, create and delete
- Systems can restrict these rights to particular users or groups of users
- Some permissions can be given to all users
- These rights can be compared to access control lists when a file is accessed
- Abstract discussion
  - Different ways to implement access control

NFS "rough edges"

- Locking
  - Inherently stateful
    - lock must persist across client calls
      - lock(), read(), write(), unlock()
  - "Separate service"
    - Handled by same server
    - Horrible things happen on server crash
    - Horrible things happen on client crash

NFS "rough edges"

- Some operations not really idempotent
  - unlink(file) returns "ok" ***once***, then "no such file"
  - server caches "a few" client requests
- Caching
  - No real consistency guarantees
  - Clients typically cache attributes, data "for a while"
  - No way to know when they're wrong

NFS "rough edges"

- Large NFS installations are brittle
  - Everybody must agree on ***many*** mount points
  - Hard to load-balance files among servers
    - No volumes
    - No atomic moves
- Cross-realm NFS access basically nonexistent
  - No good way to map uid#47 from an unknown host

Looking Ahead

- Monday
  - Inverted Index Project due
  - Timing/Coordination
- Wed – Fri: Exam