

Distributed File Systems

Amin Vahdat

CSE 123b

May 18, 2006

Annoucements

- Third assignment available today
 - Due date June 9, 5 pm

Distributed File Systems

Distributed File Systems

- Goal: transparent access to remote files
 - Access remote files as if they were stored on local hard drive
 - Why would you want this?
 - What are some of the hard issues?
- Examples:
 - NFS: Sun's Network File System
 - AFS: Andrew File System
 - Coda: CMU research project for mobile clients (now available in Linux)
 - xFS: Berkeley research project stressing "serverless" design
 - GFS: Google File System
 - FarSite: Microsoft Research project leveraging desktop drives

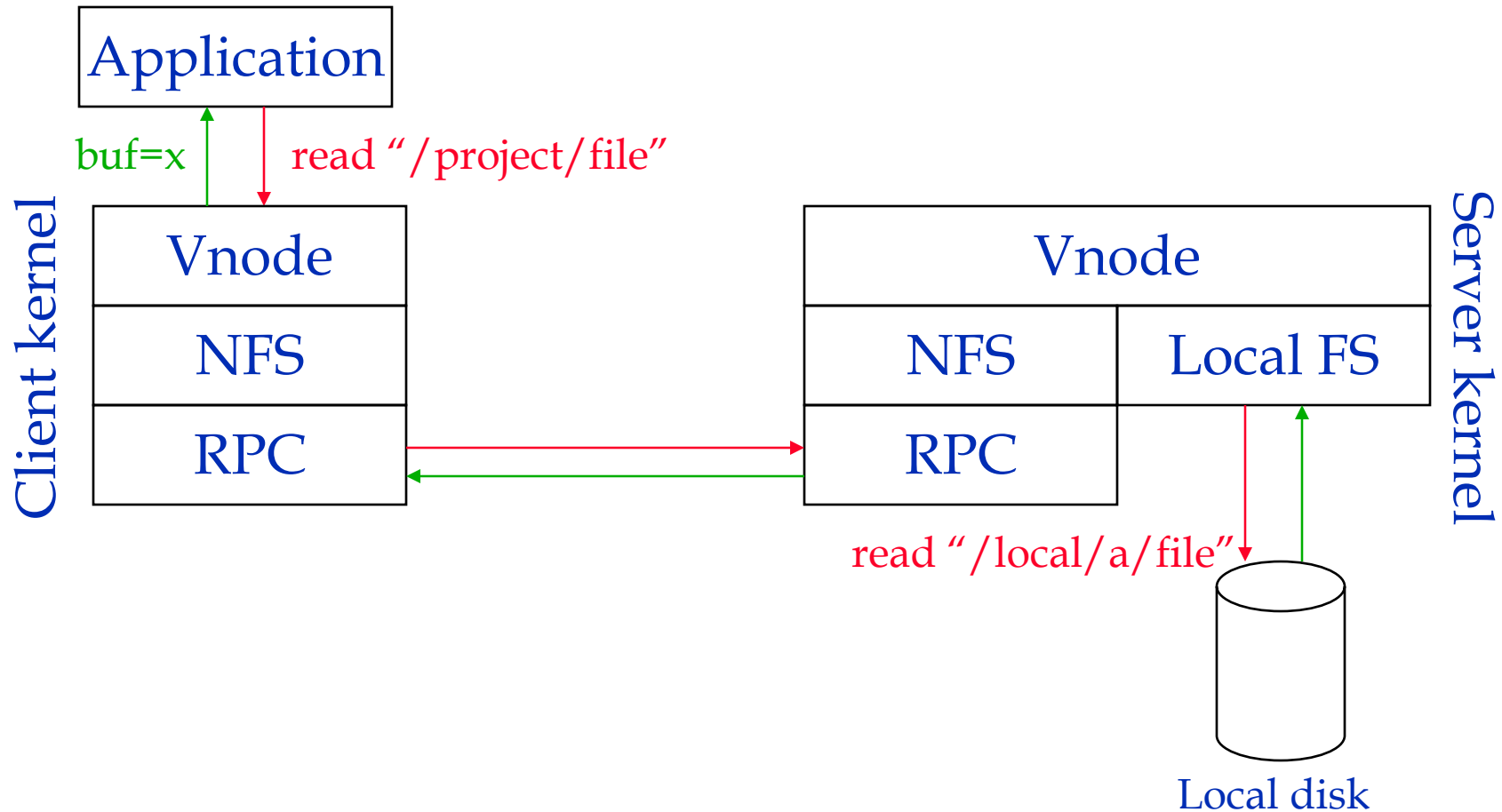
Distributed File Systems: Motivation

- Centralized administration
 - E.g., upgrades, backups, additional storage
- Same file system independent of physical machine
 - Important distributed system mantra: *location independence*
- Incremental scalability
 - Don't give everyone 20 GB disk if average user needs 1 GB
 - Add disks to central server rather than desktops

Distributed File System Issues

- Semantic transparency and performance transparency:
 - Naming: Don't change file names in moving from machine to machine
 - Caching: approximate local performance
 - Availability: remote server crash (fate sharing)
 - Security: protect sensitive information
 - What happens when you have root on file server?
 - Scale: how large can system grow?
 - In terms of storage and user base

Simplified Access Model Example



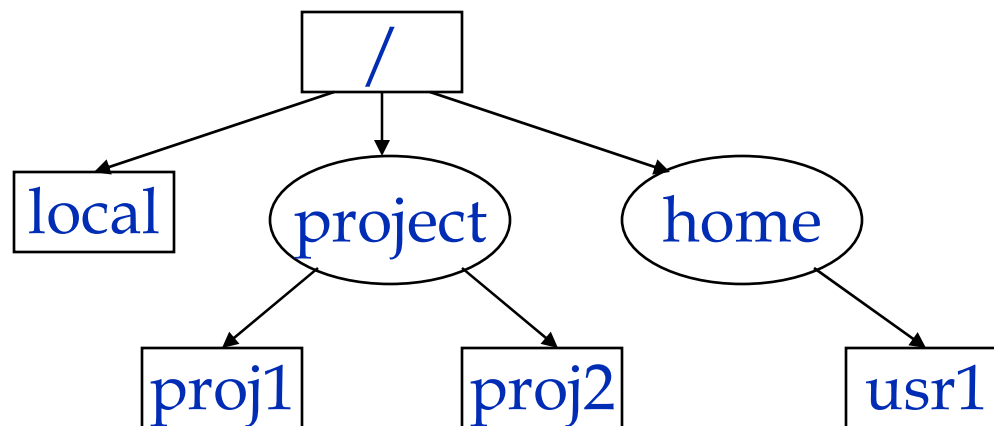
Performance

- How to make distributed file access approximate the performance of local file access?

Performance

- Network latency and limited bandwidth make it difficult to match local performance
 - But network bandwidth is surpassing disk bandwidth
 - Storage area networks, iSCSI
- How to make distributed file access approximate the performance of local file access?
 - Caching: take advantage of locality
 - Both spatial and temporal*
 - What issues are introduced by caching?

Distributed File System Structure



- Perform *mount* operation to attach remote file system into local namespace
 - E.g., /project/proj1 actually a file on remote machine (maps to server.cs.ucsd.edu:/local/a/project/proj1)

UNIX File Usage

- Most files are small ($< 10k$)
- Reads outnumber writes ($\sim 6:1$)
- Sequential access is common
- Files remain open for short period of time
 - 75% $< .5s$, 90% $< 10s$
- Most files accessed by exactly one user
 - Most shared files written by exactly one user
- Temporal locality: recently accessed files likely to be accessed again in near future
- Most bytes/files are short lived

Building a Distributed File System

- Debate in late 1980's, early 1990's:
 - Stateless vs. stateful file server
- 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
 - Where directory/file has been modified

Server Caching

- Cache read results, writes, directory operations
- Write-through vs. write-back cache?
- Pros/cons?

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

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)

Lots of 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
 - However, 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
- More recent versions of NFS use DES-based mutual authentication to provide a higher level of security
 - File data in RPC packets is not encrypted → NFS is still vulnerable

Andrew File System (AFS)

- Originally research project at CMU
 - Then, defunct commercial product from Transarc (IBM)
Public versions now available
 - Goal: single namespace for global files (sound familiar?)
- Example of stateful approach
 - Keeps track of clients accessing files
Uses callback-based invalidation
 - Eliminates need for timestamp checking, increase scalability
 - Complicates design

AFS Model

- Files cached on client disk through whole file transfer
 - Idea: migrate files to client machines where used
- Server provides *callback* promise
 - On file retrieval, server provides a time period for lease
 - Client can only access file while promise valid
- On writes, new file version flushed to server on close
 - Server *invalidates* cache of all clients that have callbacks
 - Last writer wins
- Fault-tolerance
 - Client failure: discard callbacks (soft state)
 - Server failure: server *must* remember all callbacks

AFS Semantics

- Read sharing
 - On file open, get current file version
 - On file close after update, create new version, invalidate appropriate client caches
- Write sharing
 - Last writer wins
 - Undefined semantics
 - OK for UNIX?
- Performance
 - Callbacks much more efficient than timestamp checking
 - AFS scales to thousands of hosts

Coda

- Offshoot of AFS
 - Observation: AFS does the work of migrating popular/necessary files to your machine on access
 - Nice model for mobile clients who are often disconnected
 - Use file cache to make *disconnection* transparent
 - At home, on the road, away from network connection
- Coda supplements AFS file cache with user preferences
 - E.g., always keep this file in the cache
 - Supplement with system learning user behavior
- Consistency issues?

Coda Consistency

- How to keep cached copies on disjoint hosts consistent?
 - In mobile environment, “simultaneous” writes can be separated by hours/days/weeks
- Callbacks cannot work since no network connection is available
- Coda approach (in order):
 - Assume that write sharing the rare case
 - Attempt automatic patch
 - Fallback to manual (user) intervention
- How to efficiently reintegrate changes back to main vol.