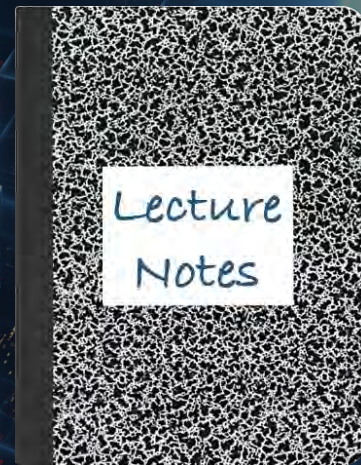


CS 417 – DISTRIBUTED SYSTEMS

Week 6: Network Attached Storage

Part 1: Design Considerations



Paul Krzyzanowski

© 2023 Paul Krzyzanowski. No part of this content may be reproduced or reposted in whole or in part in any manner without the permission of the copyright owner.

Accessing files

File sharing with socket-based programs

HTTP, FTP, sftp, telnet, ssh:

- Explicit access
- User-directed connection to access remote resources
 - Usually, a remote shell or a download of a remote file

We want more transparency

- Allow user to access remote resources just as local ones
- ⇒ **NAS: Network Attached Storage**

System Design Issues

- **Transparency**
 - Integrated into OS or access via APIs?
- **Consistency**
 - What happens if more than one user accesses the same file?
 - What if files are replicated across servers?
- **Security**
 - The local OS is no longer in charge
- **Reliability**
 - What happens when the server or client dies?
- **State**
 - Should the server keep track of clients between requests?

File service models

Download/Upload model

- *Read file*: copy file from server to client
- *Write file*: copy file from client to server

Advantage

- Simple
- Local access speeds

Problems

- **Wasteful**: what if client needs small piece?
- **Problematic**: what if client doesn't have enough space?
- **Consistency**: what if others need to modify the same file?

Remote access model

File service provides functional interface:

- *create, delete, read bytes, write bytes, etc...*

Advantages

- Client gets only what's needed
- Server can manage coherent view of file system

Problem

- Possible server and network **congestion**
 - Servers are accessed for duration of file access
 - Same data may be requested repeatedly

Semantics of file sharing

Sequential Semantics

Read returns result of last write

Easily achieved *if*

- We use a remote access model
- Server data is not replicated
- Clients do not cache data

BUT

- Performance problems if no cache
 - Clients get obsolete data
- We can **write-through**
 - Must notify all clients holding copies
 - Requires extra state, generates extra traffic

Session Semantics

Relax the rules

- Changes to an open file are visible only to the process (or machine) that modified it – made visible to others after the file is closed
- Last process to close the file overwrites any earlier changes
- To be safe – need to lock a file that is being modified to keep other clients from modifying it

Remote File Service – Components

Server

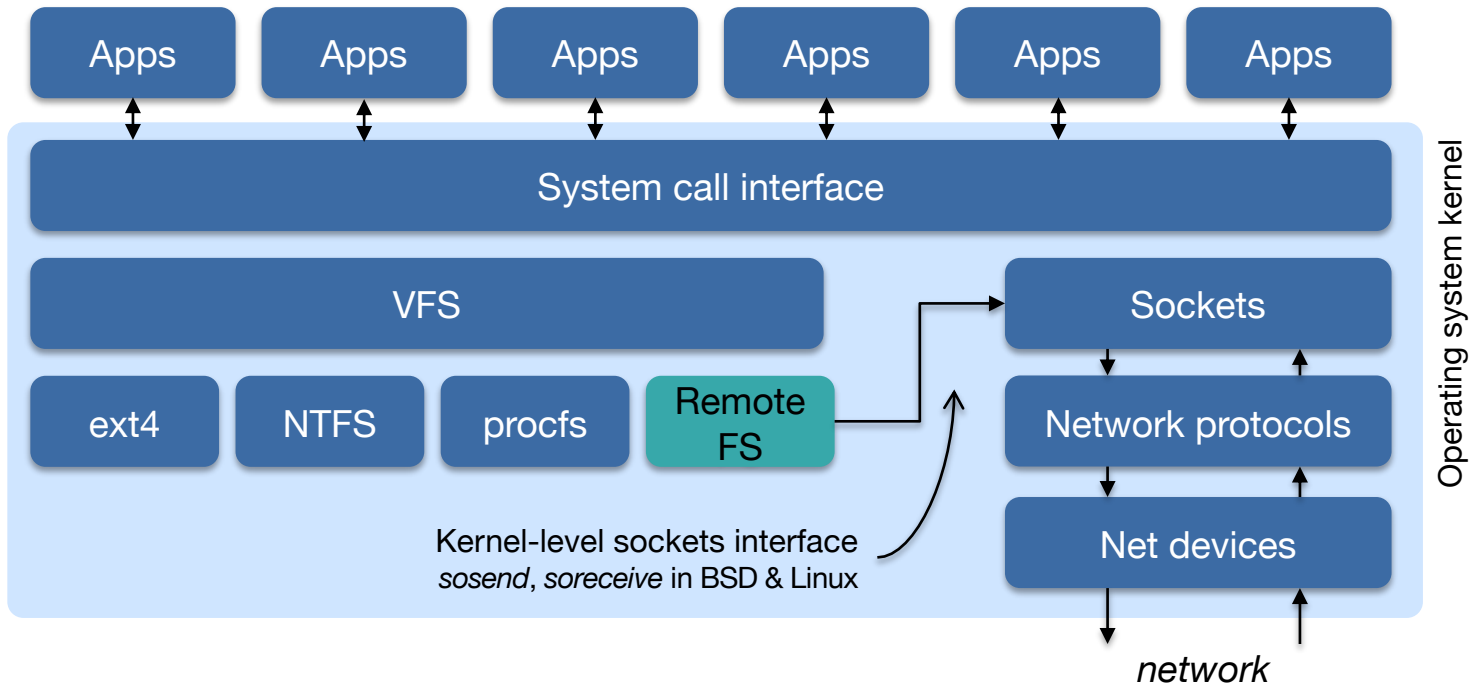
- **File Directory Service**
 - Maps textual names for file to internal locations that can be used by file service
- **File service**
 - Provides file access interface to clients

Client

- **Client module** (driver)
 - Client-side interface for the file and directory service
 - Can provide access transparency if implemented in the kernel

Accessing Remote Files

For maximum transparency, implement the client module as a file system type under VFS



Stateful or Stateless design?

Stateful

Server maintains client-specific state

- Shorter requests
- Better performance in processing requests
- Cache coherence is possible
 - Server can know who's accessing what
- File locking is possible

Stateless

Server stores no information on client accesses

- Each request must identify file and offsets
- Server can crash and recover – or fail over
 - No state to lose
- Client can crash and recover
- No open/close operations needed
 - They only establish state
- No server space used for state
 - Don't worry about the # of clients to support
- Client caching can affect consistency
- Problems if file is deleted on server
- File locking not possible

Caching

Hide latency to improve performance for repeated accesses

File data can reside in several places

- Server's disk ← *original version of the file*
- Server's buffer cache ← *memory-resident copies of recent blocks of the file*
- Client's buffer cache
- Client's disk

WARNING:

risk of cache consistency problems if multiple systems access the same file

Approaches to caching

Write-through

- What if another client reads its own (out-of-date) cached copy?
- All accesses will require checking with server
- Or ... server maintains state and sends invalidations

Delayed writes (write-behind)

- Data can be buffered locally (watch out for consistency – others won't see updates!)
- Remote files updated periodically
- One bulk write is more efficient than lots of little writes
- Problem: semantics become ambiguous

Write on close

- Admit that we have session semantics

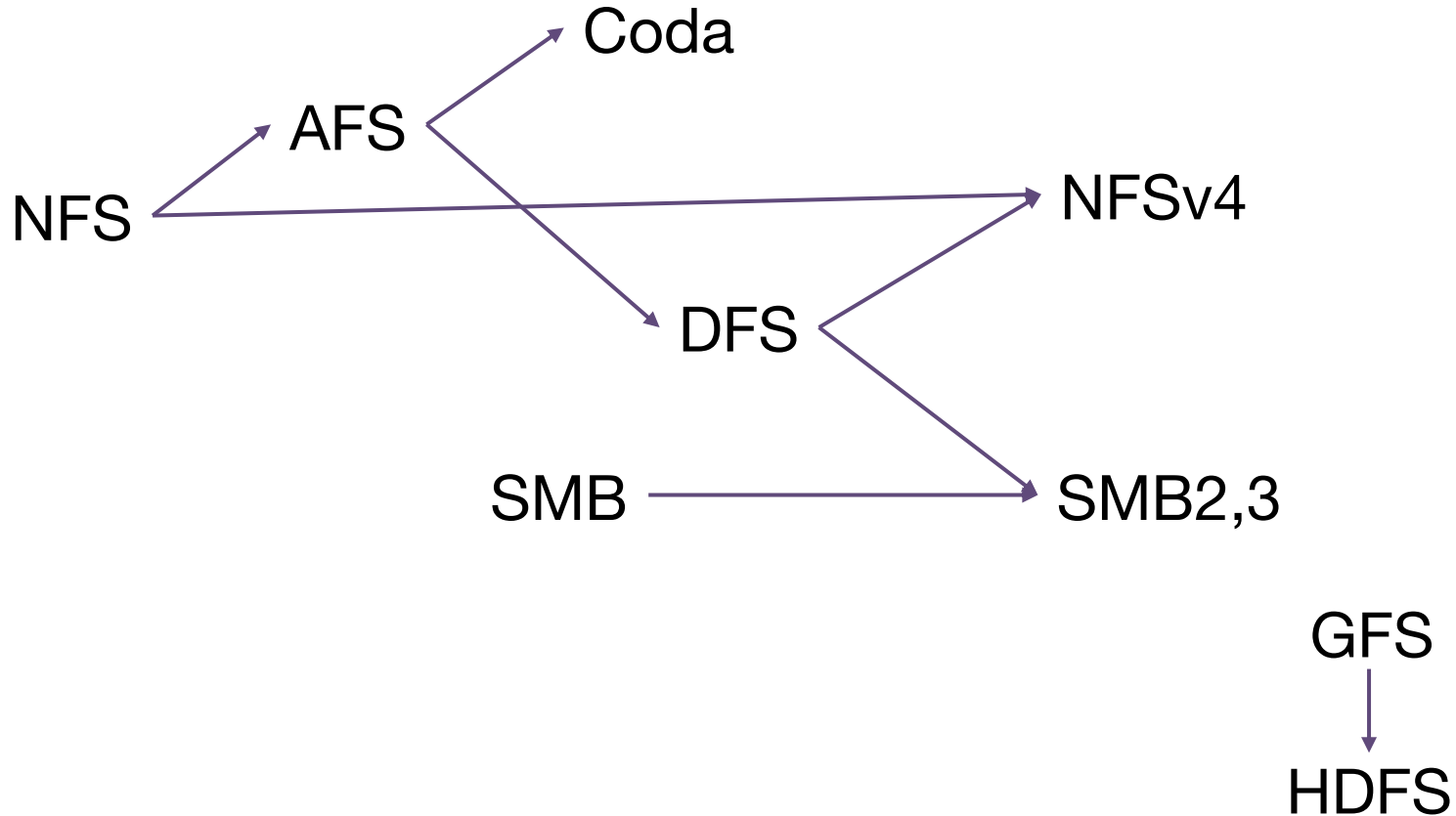
Read-ahead (prefetch)

- Request chunks of data before it is needed
- Minimize wait times if that data is later needed

Centralized control

- Keep track of who has what open and cached on each node
- More state to track on the server & more messages

Next...



The End