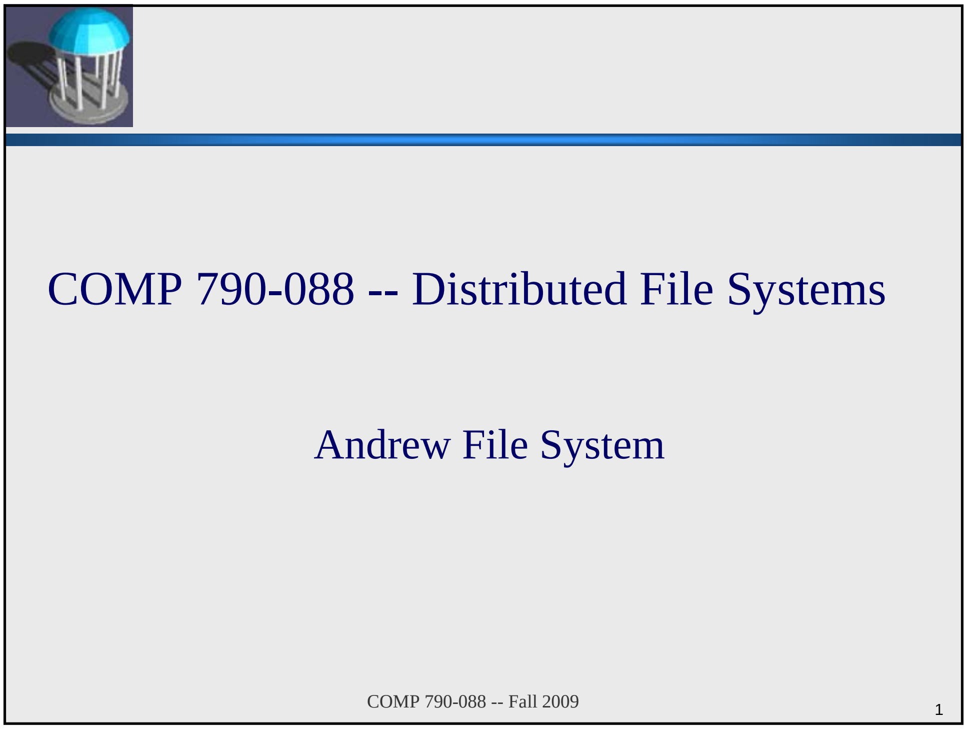
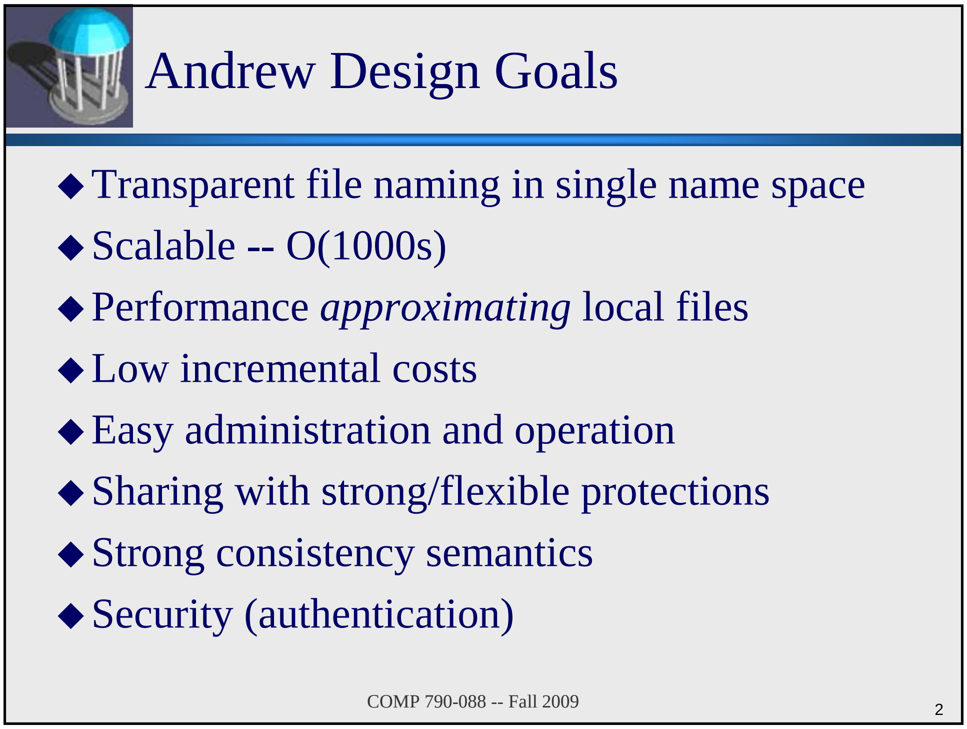


COMP 790-088 -- Distributed File Systems

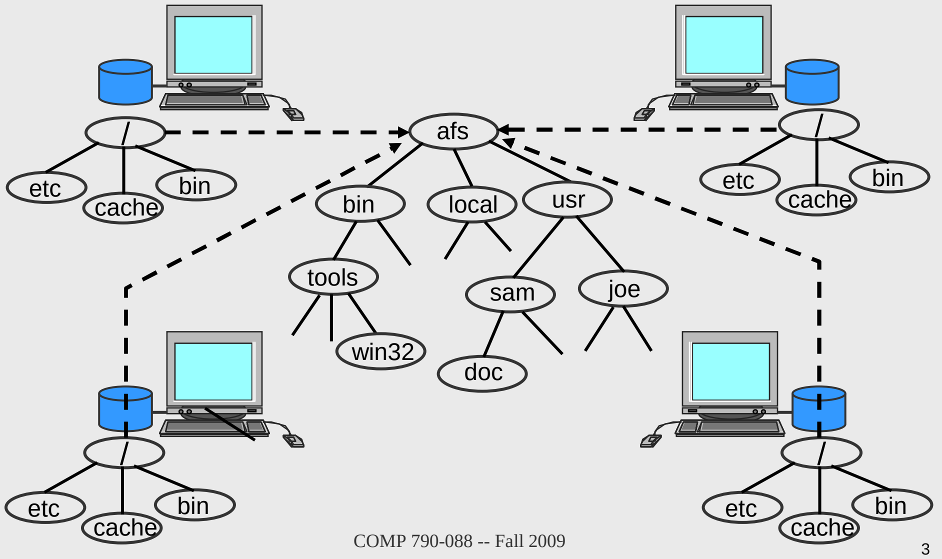
Andrew File System



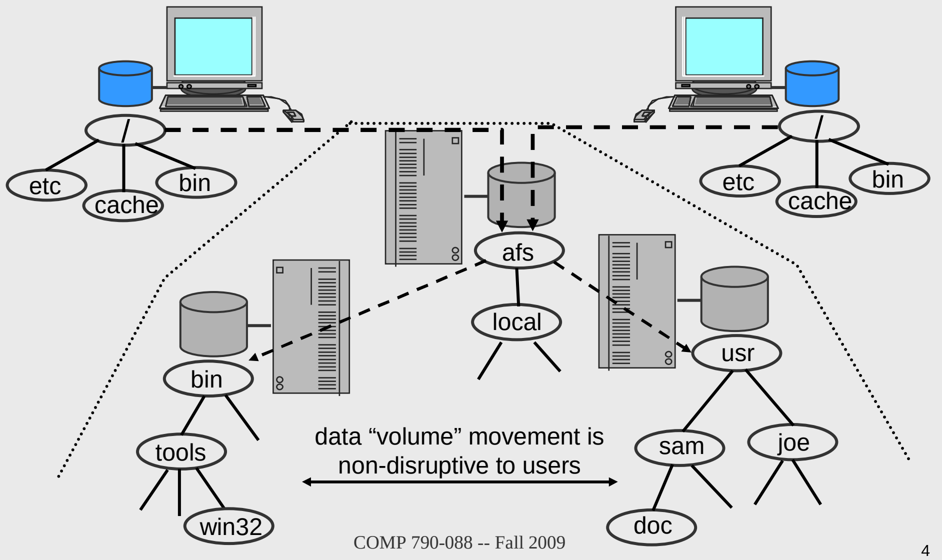
Andrew Design Goals

- Transparent file naming in single name space
- Scalable -- O(1000s)
- Performance *approximating* local files
- Low incremental costs
- Easy administration and operation
- Sharing with strong/flexible protections
- Strong consistency semantics
- Security (authentication)

Users Share a Single File System

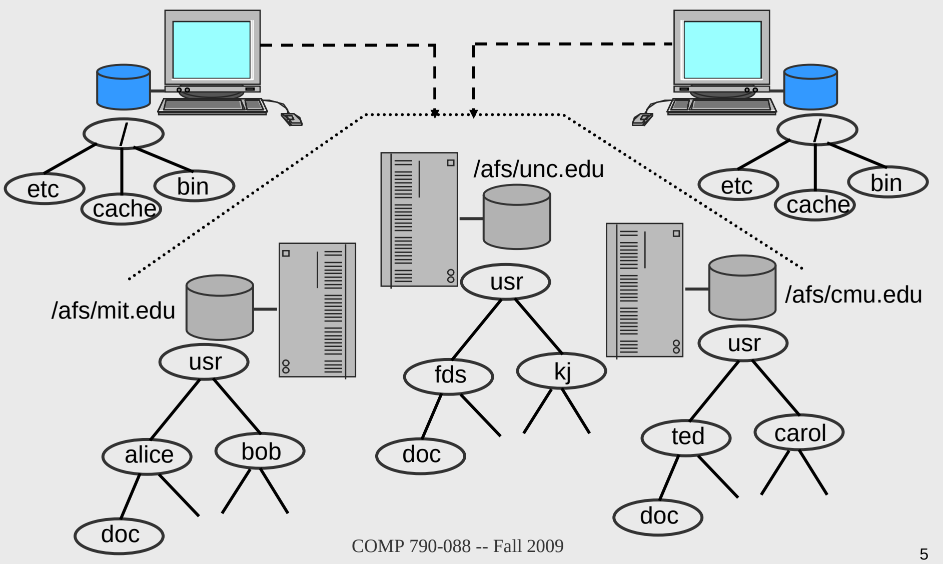


File Space Distributed on Servers



data “volume” movement is non-disruptive to users

Server "Cells" Form a Global File System



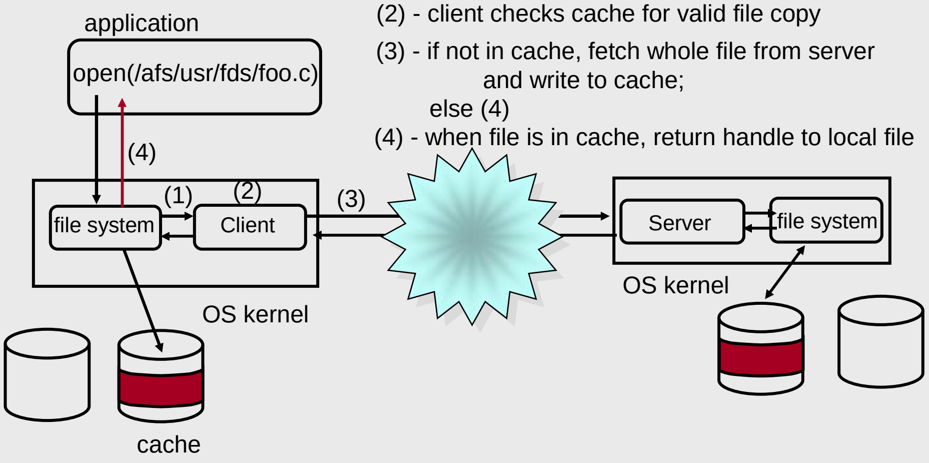
Andrew File Access (Open)

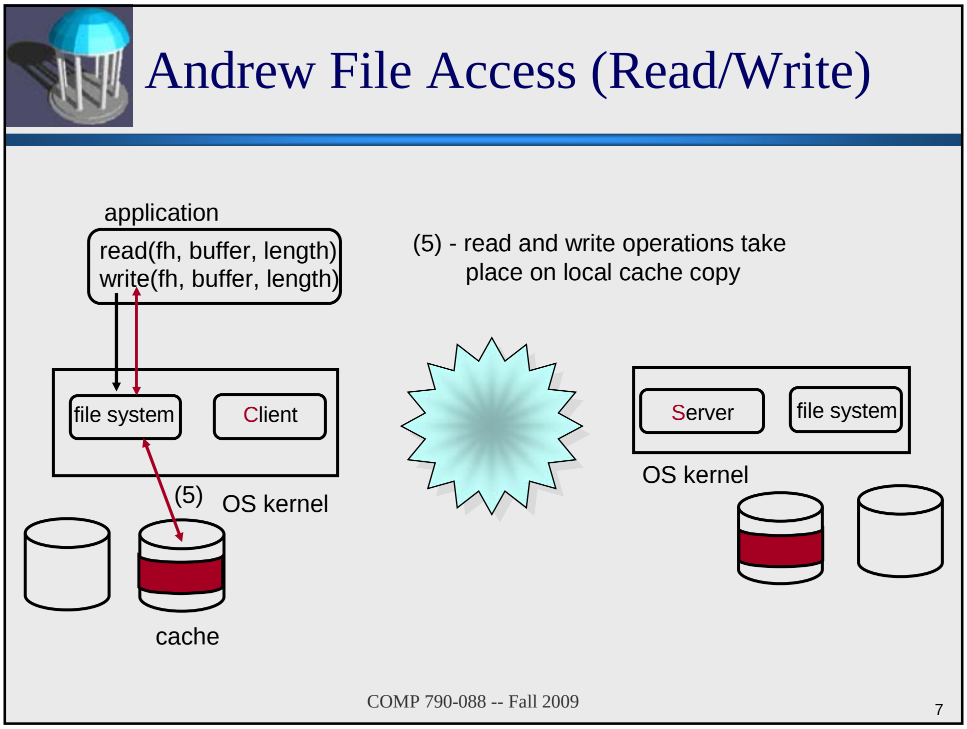
(1) - open request passed to Andrew client

(2) - client checks cache for valid file copy

(3) - if not in cache, fetch whole file from server
and write to cache;
else (4)

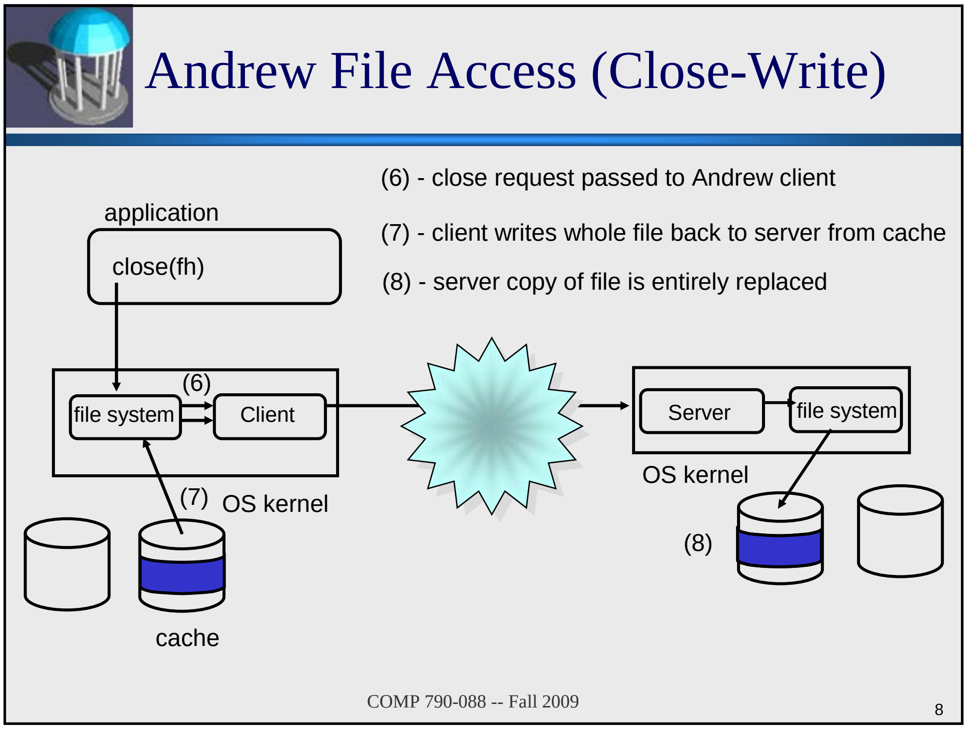
(4) - when file is in cache, return handle to local file





Andrew File Access (Read/Write)

(5) - read and write operations take place on local cache copy



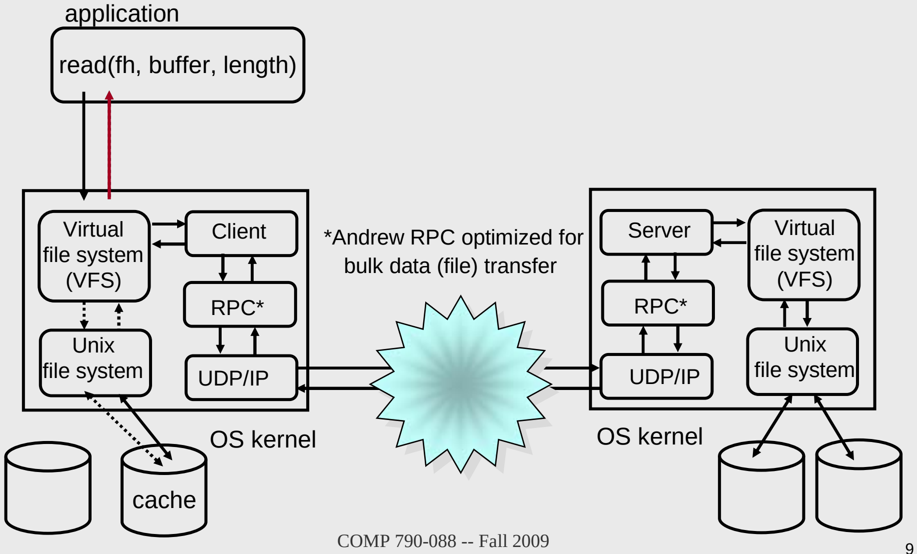
Andrew File Access (Close-Write)

(6) - close request passed to Andrew client

(7) - client writes whole file back to server from cache

(8) - server copy of file is entirely replaced

Andrew System Structure



application

read(fh, buffer, length)

Virtual file system (VFS)

Client

RPC*

Unix file system

UDP/IP

OS kernel

cache

*Andrew RPC optimized for bulk data (file) transfer

Server

Virtual file system (VFS)

RPC*

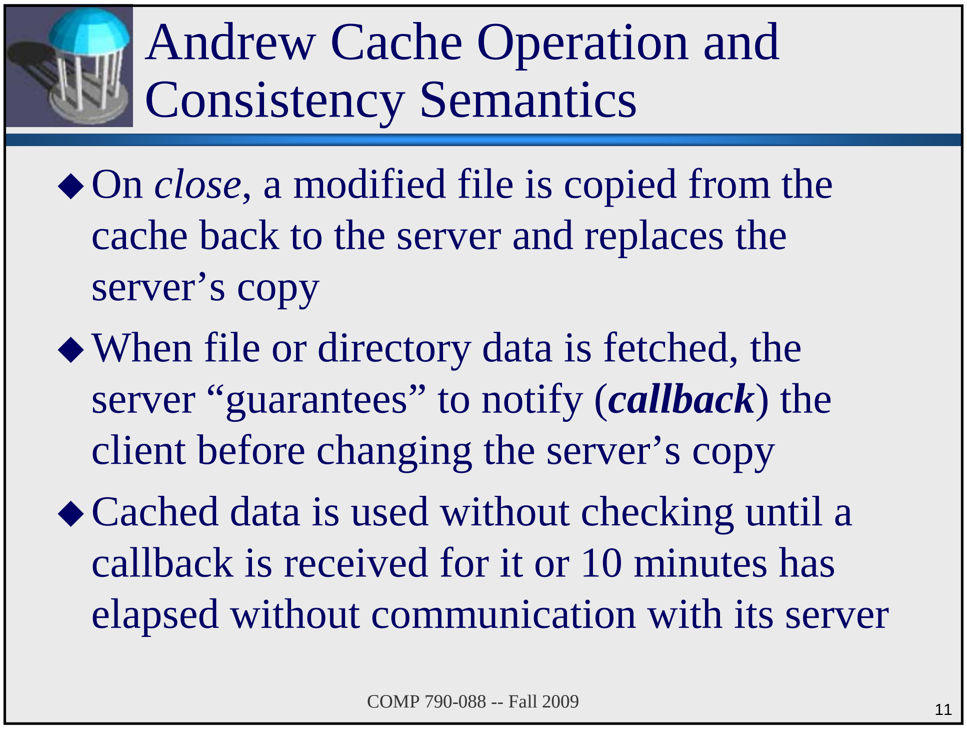
UDP/IP

Unix file system

OS kernel

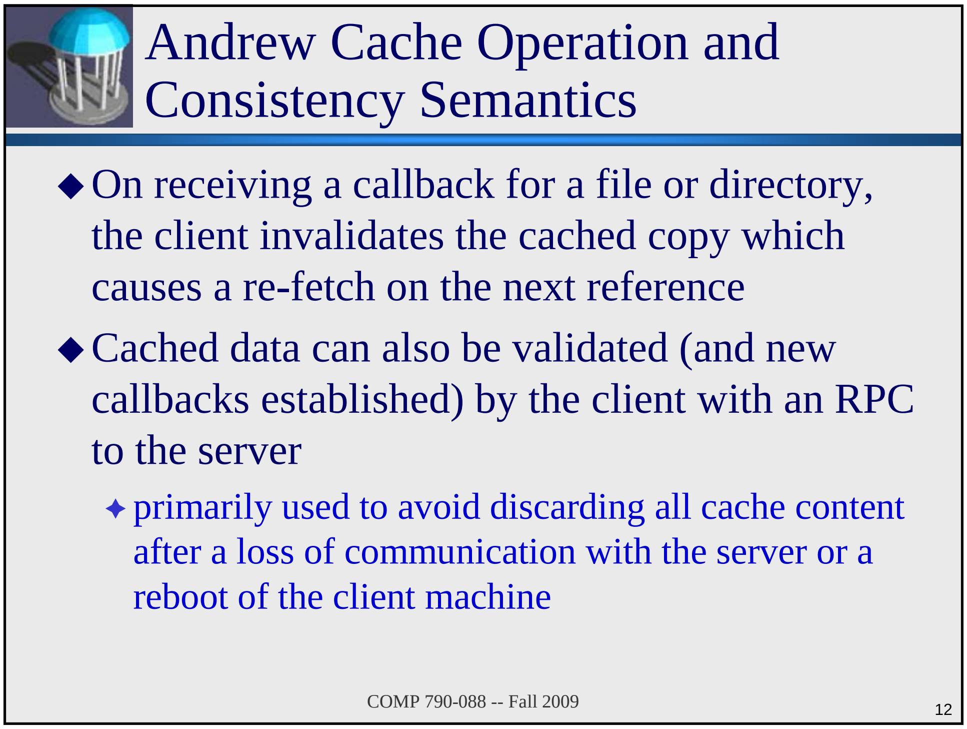
Andrew Cache Operation and Consistency Semantics

- File data is read from, or written to, the local cache copy only
  - if the file exists but is not in the cache, it must be fetched from the server to the local cache on *open*
- Directories are fetched to the local cache for read (*lookup*) operations
- Directory updates (e.g., *create* or *remove*) are always performed on the server (atomically) and also performed locally on a cached copy if present and valid



Andrew Cache Operation and Consistency Semantics

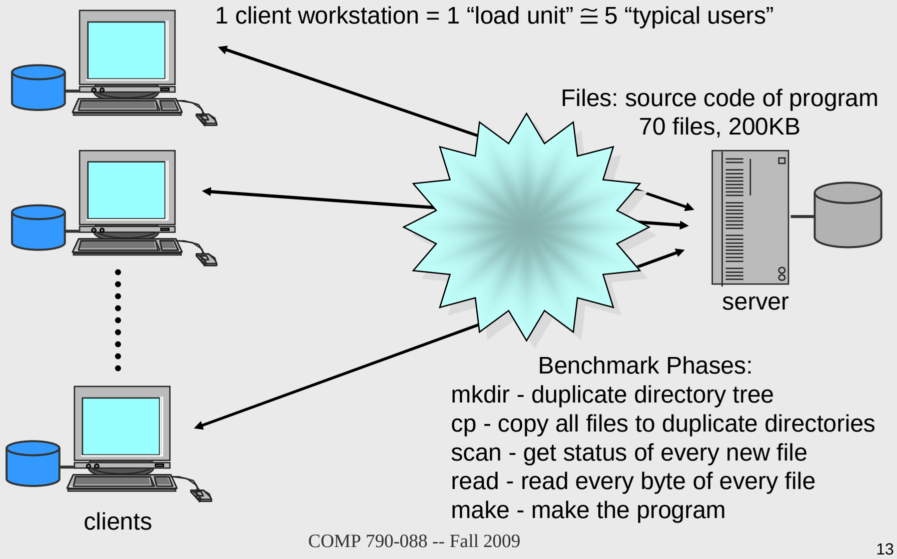
- On *close*, a modified file is copied from the cache back to the server and replaces the server's copy
- When file or directory data is fetched, the server "guarantees" to notify (***callback***) the client before changing the server's copy
- Cached data is used without checking until a callback is received for it or 10 minutes has elapsed without communication with its server



Andrew Cache Operation and Consistency Semantics

- On receiving a callback for a file or directory, the client invalidates the cached copy which causes a re-fetch on the next reference
- Cached data can also be validated (and new callbacks established) by the client with an RPC to the server
  - primarily used to avoid discarding all cache content after a loss of communication with the server or a reboot of the client machine

Andrew Benchmark



1 client workstation = 1 “load unit” ≅ 5 “typical users”

Files: source code of program
70 files, 200KB

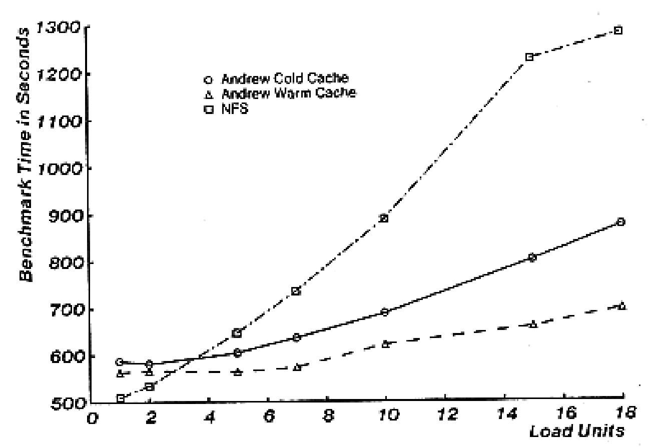
server

clients

Benchmark Phases:
mkdir - duplicate directory tree
cp - copy all files to duplicate directories
scan - get status of every new file
read - read every byte of every file
make - make the program

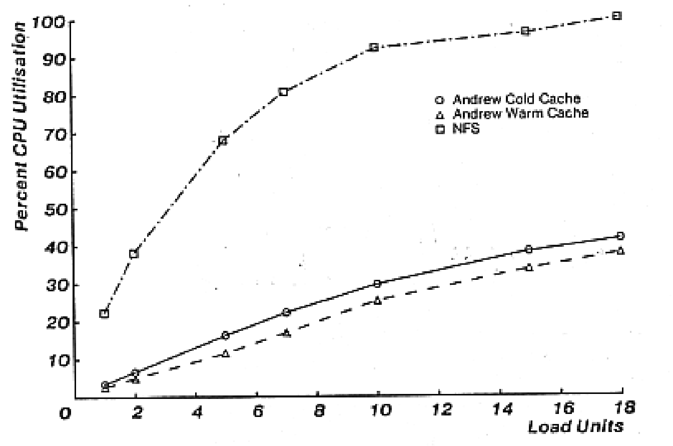
Andrew Benchmark -- Elapsed Time

Source: Howard, et al, “Scale and Performance in a Distributed File System”, ACM TOCS, vol. 6, no. 1, February 1988, pp. 51-81.



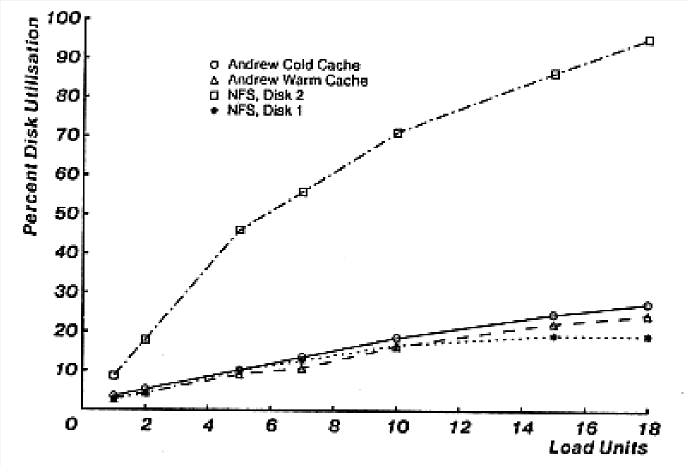
Andrew Benchmark -- Server CPU Utilization

Source: Howard, et al, “Scale and Performance in a Distributed File System”, ACM TOCS, vol. 6, no. 1, February 1988, pp. 51-81.



Andrew Benchmark -- Server Disk Utilization

Source: Howard, et al, “Scale and Performance in a Distributed File System”, ACM TOCS, vol. 6, no. 1, February 1988, pp. 51-81.


Percent Disk Utilisation
Load Units
○ Andrew Cold Cache
△ Andrew Warm Cache
□ NFS, Disk 2
✱ NFS, Disk 1