

**CSE 451: Operating Systems
Winter 2004**

**Module 22
Distributed File Systems**

Ed Lazowska
lazowska@cs.washington.edu
Allen Center 570

Distributed File Systems

- The most common distributed services:
 - printing
 - email
 - files
- Basic idea of distributed file systems: support network-wide sharing of files and devices (disks)
- Generally provide a “traditional” view – a centralized shared file system – but with a distributed implementation

3/7/2004

© 2004 Ed Lazowska & Hank Levy

2

Issues

- What is the basic abstraction
 - remote file system?
 - open, close, read, write, ...
 - remote disk?
 - read block, write block
- Naming
 - how are files named?
 - are those names **location transparent**?
 - is the file location visible to the user?
 - are those names **location independent**?
 - do the names change if the file moves?
 - do the names change if the user moves?

3/7/2004

© 2004 Ed Lazowska & Hank Levy

3

- Caching
 - caching exists for performance reasons
 - where are file blocks cached?
 - on the file server?
 - on the client machine?
 - both?
- Sharing and coherency
 - what are the semantics of sharing?
 - what happens when a cached block/file is modified
 - how does a node know when its cached blocks are out of date?

3/7/2004

© 2004 Ed Lazowska & Hank Levy

4

- Replication
 - replication can exist for performance and/or availability
 - can there be multiple copies of a file in the network?
 - if multiple copies, how are updates handled?
 - what if there's a network partition and clients work on separate copies?
- Performance
 - what is the cost of remote operation?
 - what is the cost of file sharing?
 - how does the system scale as the number of clients grows?
 - what are the performance limitations: network, CPU, disks, protocols, data copying?

3/7/2004

© 2004 Ed Lazowska & Hank Levy

5

Example: SUN Network File System (NFS)

- The Sun Network File System (NFS) has become a common standard for distributed UNIX file access
- NFS runs over LANs (even over WANs – slowly)
- Basic idea: allow a remote directory to be “mounted” (spliced) onto a local directory, giving access to that remote directory and all its descendants as if they were part of the local hierarchy
- Pretty much exactly like a “local mount” or “link” on UNIX
 - except for implementation and performance ...
 - no, we didn't really learn about these, but they're obvious ☺

3/7/2004

© 2004 Ed Lazowska & Hank Levy

6

- For instance:
 - I mount `/u4/lazowska` on Node1 onto `/students/fo` on Node2
 - users on Node2 can then access this directory as `/students/fo`
 - if I had a file `/usr/lazowska/myfile`, users on Node2 see it as `/students/fo/myfile`
- Just as, on a local system, I might link `/homes/gws/lazowska/www` as `/u4/lazowska/mywww` to allow easy access to my web data from my home directory

3/7/2004

© 2004 Ed Lazowska & Hank Levy

7

NFS implementation

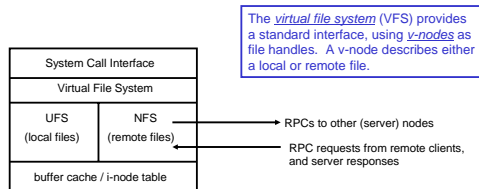
- NFS defines a set of RPC operations for remote file access:
 - searching a directory
 - reading directory entries
 - manipulating links and directories
 - reading/writing files
- Every node may be both a client and server

3/7/2004

© 2004 Ed Lazowska & Hank Levy

8

- NFS defines new layers in the Unix file system



3/7/2004

© 2004 Ed Lazowska & Hank Levy

9

NFS caching / sharing

- On an open, the client asks the server whether its cached blocks are up to date.
- Once a file is open, different clients can write it and get inconsistent data.
- Modified data is flushed back to the server every 30 seconds.

3/7/2004

© 2004 Ed Lazowska & Hank Levy

10

Example: CMU's Andrew File System (AFS)

- Developed at CMU to support all of its student computing
- Consists of workstation clients and dedicated file server machines (*differs from NFS*)
- Workstations have local disks, used to cache files being used locally (originally whole files, now 64K file chunks) (*differs from NFS*)
- Andrew has a single name space – your files have the same names everywhere in the world (*differs from NFS*)
- Andrew is good for distant operation because of its local disk caching: after a slow startup, most accesses are to local disk

3/7/2004

© 2004 Ed Lazowska & Hank Levy

11

AFS caching/sharing

- Need for scaling required reduction of client-server message traffic
- Once a file is cached, all operations are performed locally
- On close, if the file has been modified, it is replaced on the server
- The client assumes that its cache is up to date, unless it receives a *callback* message from the server saying otherwise
 - on file open, if the client has received a callback on the file, it must fetch a new copy; otherwise it uses its locally-cached copy (*differs from NFS*)

3/7/2004

© 2004 Ed Lazowska & Hank Levy

12

Example: Berkeley Sprite File System

- Unix file system developed for *diskless* workstations with large memories at UCB (*differs from NFS, AFS*)
- Considers memory as a huge cache of disk blocks
 - memory is shared between file system and VM
- Files are permanently stored on servers
 - servers have a large memory that acts as a cache as well
- Several workstations can cache blocks for read-only files
- If a file is being written by more than 1 machine, client caching is turned off – all requests go to the server (*differs from NFS, AFS*)

3/7/2004

© 2004 Ed Lazowska & Hank Levy

13

Summary

- There are a number of issues to deal with:
 - what is the basic abstraction
 - naming
 - caching
 - sharing and coherency
 - replication
 - performance
- No right answer! Different systems make different tradeoffs!

3/7/2004

© 2004 Ed Lazowska & Hank Levy

14

- Performance is always an issue
 - always a tradeoff between performance and the semantics of file operations (e.g., for shared files).
- Caching of file blocks is crucial in any file system
 - maintaining coherency is a crucial design issue.
- Newer systems are dealing with issues such as disconnected operation for mobile computers

3/7/2004

© 2004 Ed Lazowska & Hank Levy

15