Distributed file systems, Case studies

- Sun's NFS
  - history
  - virtual file system and mounting
  - NFS protocol
  - caching in NFS
  - V3
- Andrew File System
  - history
  - organization
  - caching
  - DFS
- AFS vs. NFS

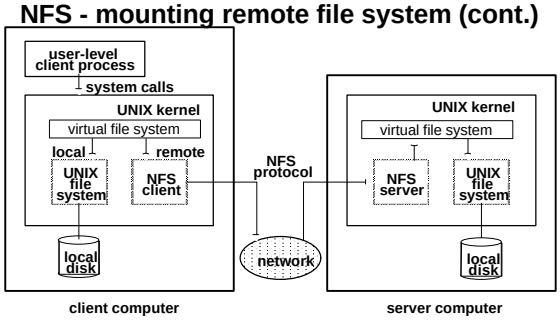
Sun's network file systems (NFS)

- Designed by Sun Microsystems
  - First distributed file service designed as a project, introduced in 1985
  - To encourage its adoption as a standard
    - Definitions of the key interfaces were placed in the public domain in 1989
    - Source code for a reference implementation was made available to other computer vendors under license
    - Currently the de facto standard for LANs
- Provides transparent access to remote files on a LAN, for clients running on UNIX and other operating systems
  - A UNIX computer typically has a NFS client and server module in its OS kernel
    - Available for almost any UNIX
    - Client modules are available for Macintosh and PCs

NFS - mounting remote file system (cont.)

- Remote file systems may be
  - Hard mounted — when a user-level process accesses a file, it is suspended until the request can be completed
    - If a server crashes, the user-level process will be suspended until recovers
  - Soft mounted — after a small number of retries, the NFS client returns a failure code to the user process
    - Most UNIX utilities don't check this code…
- Automounting
  - The automounter dynamically mounts a file system whenever an "empty" mount point is referenced by a client
    - Further accesses do not result in further requests to the automounter…
    - Unless there are no references to the remote file system for several minutes, in which case the automounter unmounts it

NFS - mounting remote file system (cont.)



- Virtual file system:
  - Separates generic file-system operations from their implementation (can have different types of local file systems)
  - Based on a file descriptor called a vnode that is unique networkwide (UNIX inodes are only unique on a single file system)

NFS protocol

- NFS protocol provides a set of RPCs for remote file operations
  - Looking up a file within a directory
  - Manipulating links and directories
  - Creating, renaming, and removing files
  - Getting and setting file attributes
  - Reading and writing files
- NFS is stateless
  - Servers do not maintain information about their clients from one access to the next
    - There are no open-file tables on the server
  - There are no open and close operations
    - Each request must provide a unique file identifier, and an offset within the file
  - Easy to recover from a crash, but file operations must be idempotent

NFS protocol (cont.)

- Because NFS is stateless, all modified data must be written to the server's disk before results are returned to the client
  - Server crash and recovery should be invisible to client —data should be intact
  - Lose benefits of caching
    - Solution — RAM disks with battery backup (un-interruptable power supply), written to disk periodically
- A single NFS write is guaranteed to be atomic, and not intermixed with other writes to the same file
  - However, NFS does not provide concurrency control
    - A `write` system call may be decomposed into several NFS writes, which may be interleaved
    - Since NFS is stateless, this is not considered to be an NFS problem

Caching in NFS

- Traditional UNIX
  - Caches file blocks, directories, and file attributes
  - Uses read-ahead (prefetching), and delayed-write (flushes every 30 seconds)
- NFS servers
  - Same as in UNIX, except server's write operations perform write-through
    - Otherwise, failure of server might result in undetected loss of data by clients
- NFS clients
  - Caches results of read, write, getattr, lookup, and readdir operations
  - Possible inconsistency problems
    - Writes by one client do not cause an immediate update of other clients' caches

Caching in NFS (cont.)

- NFS clients (cont.)
  - File reads
    - When a client caches one or more blocks from a file, it also caches a timestamp indicating the time when the file was last modified on the server
    - Whenever a file is opened, and the server is contacted to fetch a new block from the file, a validation check is performed
      - Client requests last modification time from server, and compares that time to its cached timestamp
      - If modification time is more recent, all cached blocks from that file are invalidated
      - Blocks are assumed to valid for next 3 seconds (30 seconds for directories)
  - File writes
    - When a cached page is modified, it is marked as dirty, and is flushed when the file is closed, or at the next periodic flush
  - Now two sources of inconsistency: delay after validation, delay until flush

NFS v2 performance improvements, v3

- Client-side caching:
  - every NFS operation requires network access - slow. client can cache the data it currently works on to speed up access to it.
  - problem - multiple clients access the data - multiple copies of cached data may become inconsistent.
  - Solutions - cache only read-only files; cache files where inconsistency is not vital (inodes) and check consistently frequently
- deferral of writes
  - client side - the clients bunch writes together (if possible) before sending them to server (if the client crushes - it knows where to restart)
  - a sever must commit the writes to stable storage before reporting them to a client. Battery backed non-volatile memory (NVRAM) is used (rather than the disk). NVRAM -> disk transfers are optimized for disk head movements and written to disk later.
- v3 allows delayed writes by introducing commit operation - all writes are "volatile" until the server processes commit operation.
- Requires changes in semantics - applications programmer cannot assume that all the writes are done and should explicitly issue commit .

Andrew file system (AFS)

- Designed by Carnegie Mellon University
  - Developed during mid-1980s as part of the Andrew distributed computing environment
  - Designed to support a WAN of more than 5000 workstations
  - Much of the core technology is now part of the Open Software Foundation (OSF) Distributed Computing Environment (DCE), available for most UNIX and some other operating systems
- AFS was made to span large campuses and scale well therefore the emphasis was placed on offloading the work to the clients
- as much as possible data is cached on clients, uses session semantics - cache consistency operations are done when file is opened or closed
- Provides transparent access to remote files on a WAN, for clients running on UNIX and other operating systems
  - Access to all files is via the usual UNIX file primitives
  - Compatible with NFS — servers can mount NFS file systems

AFS (cont.)

- AFS is stateful, when a client reads a file from a server it holds a callback, the server keeps track of callbacks and when one of the clients closes the file (and synchronizes it's cached copy) and updates it, the server notifies all the callback holders of the change breaking the callback, callbacks can be also broken to conserve storage at server
- problems with AFS:
  - even if the data is in local cache - if the client performs a write a complex protocol of local callback verification with the server must be used; cache consistency preservation leads to deadlocks
  - in a stateful model, it is hard to deal with crushes.

Caching in Andrew

- When a remote file is accessed, the server sends the entire file to the client
  - The entire file is then stored in a disk cache on the client computer
    - Cache is big enough to store several hundred files
- Implements session semantics
  - Files are cached when opened
  - Modified files are flushed to the server when they are closed
  - Writes may not be immediately visible to other processes
- When client caches a file, server records that fact — it has a callback on the file
  - When a client modifies and closes a file, other clients lose their callback, and are notified by server that their copy is invalid

How can Andrew perform well?

- Most file accesses are to files that are infrequently updated, or are accessed by only a single user, so the cached copy will remain valid for a long time
- Local cache can be big — maybe 100 MB — which is probably sufficient for one user's working set of files
- Typical UNIX workloads:
  - Files are small, most are less than 10kB
  - Read operations are 6 times more common than write operations
  - Sequential access is common, while random access is rare
  - Most files are read and written by only one user; if a file shared, usually only one user modifies it
  - Files are referenced in bursts

DCE Distributed File System (DCE DFS)

- Modification of AFS by Open Software Foundation for its Distributed Computing Environment (DCE)
- switch to Unix file sharing semantics, implement using tokens
  - tokens provide finer degree of control than callbacks of AFS
  - tokens
    - fine grained: tokens (read/write) on portions of a file
    - type specific: open, read, write, lock, status, update
  - a client can "check out" a token from a server for a particular file. Only if a client holds a lock token the client is allowed to modify the file; if a client holds status token it is allowed to modify status of the file
  - token expires in 2 minutes (for fault-tolerance)
- replication – easy replication of filesystems on several machines, transparent to the user,
  - one server – primary does all updates
  - others – read-only

AFS vs. NFS

- NFS is simpler to implement, AFS is cumbersome due to cache consistency synchronization mechanisms, etc
- NFS does not scale well - it is usually limited to a LAN, AFS scales well - it may span the Internet
- NFS performs better than AFS under light to medium load. AFS does better under heavy load
- NFS does writes faster(?)