1 Distributed File Systems

1.1 Goals

- Network transparency
  users do not have to aware the location of files to access them
  - location transparency: the name of a file does not reveal any kind of the file's physical storage location.
  - location independence: the name of a file does not need to be changed when the file's physical storage location changes.
- High availability
  system failures or sceduled activities such as backups, addition of nodes

1.2 Architecture

- Computation model
  - file severs – machines dedicated to storing files and performing storage and retrieval operations (for high performance)
  - clients – machines used for computational activities may have a local disk for caching remote files
- Two most important services
  - name server – maps user specified name to stored objects, files and directories
  - cache manager – to reduce network delay, disk delay problem: inconsistency
- Typical data access actions (Figure 9.2)

1.3 Mechanisms for Buiding Distributed File Systems

- Mounting – binding together of different filename spaces to form a single hierarchically structured name space
- Caching – to reduce delays in the accessing of data, temporal locality
- Hints – treat cached data as hints
- Bulk Data Transfer – to reduce protocol processing overhead
- Encryption – to enforce security

1.4 Design Issues

1.4.1 Naming and Name Resolution

- name resolution – mapping a name to an object
- a name space – collection of names
- 3 traditional ways to name files in a distributed environment
  - concated the host name to the names of files stored on that host
    system-wide uniquess guaranteed;
    however, not network transparent, not location independent,
    simple to located a file
  - mount remote directories onto local directories,
    once mounted, files can be referenced in a location-transparent manner
  - provide a single global directory,
    requires a unque file name for every file,
    cannot encompass heterogenous environments and wide geograpical areas

1.4.2 Caches on Disk or Main Memory

- client-cache in main memory
  can be used by diskless workstation
  faster to access from main memory than disk
  compete with the virtual memory system for physical memory space
- client-cache on a local disk
  large files can be cached
  the virtual memory management is simpler
  a workstation can function even when it is disconnected from the network

1.4.3 Writing Policy

- write-through – all writes are carried out immediately
  reliable – little information is lost in the event of a client crash
  slow – cache not that useful
- delayed-write – delays writing at the server
  possible perform many writes to a block in the cache before it is written
  if data is written and then deleted immediately, data need not be written at all (20-30 % of new data is deleted with 30 secs)
- write-on-close – delay writing until the file is closed at the client
  if file is open for short, works fine but not much benefit over delayed-write
  if file is open for long, susceptible to losing data in the event of a client crash

1.4.4 Consistency

Consistency Semantics (pp. 375-376, Silverschatz)
Assume open; reads/writes; close

- UNIX semantics
  - Writes to an open file are visible immediately to others that have this file opened at the same time.
- Session semantics
  - Writes to an open file by a user is not visible immediately by othe users that have files opened already
  - Once a file is closed, the changes made by it are visible by sessesion started later.
- Immutable-Shared-Files semantics
  - A sharable file cannot be modified.
  - File names cannot be reused and its contents may not be altered.
  - Simple to implement.

How to ensure that a locally cached data is consistent with the master data?

1. Client-initiated approach – check validity on
   every access: too much overhead
   first access to a file (e.g., file open)
   every fixed time interval
2. Server-initiated approach – server records, for each client, the (parts of) files it caches. After the server detects a potential inconsistency, it reacts. Potential inconsistency:
   In session semantics, a client closes a modified file.
   In UNIX sematics, the server must be notified whenever a file is opened and the intended mode (read or write mode) must be indicated for every open.
   Disable cache when a file is opened in conflicting modes.
3. Not allow caching when concurrent-write sharing occurs
   Allow many readers
   If a client opens for writing, inform all the clients to purge their cached data.

1.4.5 A Comparison of Caching and Remote Service

1. reduces remote accesses (esp, when locality is capitalized) $\Longrightarrow$ reduces network traffice and server load
2. total network overhead is lower for big chunks of data (caching) than a series of reponses to specific requests.
3. disk access can be optimized better for large requests than random disk blocks

4. cache-consistency problem is the major drawback. If there are frequent writes, overhead due to the consistency problem is significant.

5. OS is simipler for remote service.

1.4.6 Stateful versus Stateless Service

two approaches to server-side information

1. stateful file server
   a client performs open on a file
   the server keeps file information (e.g., file descriptor entry, offset)
   Adv: increased performance
   On server crash, it looses all its volatile state information
   On client crash, the server needs to know to claim state space

2. stateless file server – each request is self-contained
   each request identifies the file, the position, read/write.
   server failure is identical to slow server (client retries...)
   each request must be idempotent.
   NFS employes this.

1.4.7 Availability

Replication – enhances the availability of files

- Issues:
  1. how to keep the replicas of a file consistent
  2. how to detect inconsistencies among replicas of a cause and later recover from these inconsistencies
- Problems – mainly with updates
  1. a replica is not updated due to its server failure
  2. network partitioned
- Replication Management
  1. weighted vote for read and write
  2. current synchronization site for each file group to control access

1.4.8 Scalability

the design of a system should cater the demands of a growing system

1.4.9 Semantics

Read should return the data (stored) due to the latest write operation – difficult and expensive to gurantee

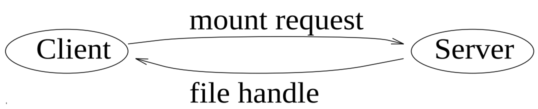
2 The Sun Network File System

- NFS views a set of interconnected workstations as a set of independent machine with independent file systems, 1985
- Goal: allow sharing among these files (or explicit request) in a transparent manner.
- Server-Client relationship between any pair of machines.
- mount remote directory on a client's machine

2.1 The Mount Protocol

used to establish the initial logical connection between a server and a client.

1. the server maintains an export list

   What is in file handle? – fields identifying the file system type, the disk, the i-node number of the directory, security information.

2. the server maintains a list of the client machines and the corresponding currently mounted directories.
3. static mounting – `/etc/rc`, a shell script containing the remote mount commands.
4. automounting – server contacted.

   the first time a remote file is opened, the operating system send a message to each of the servers. The first one to reply wins, and its directory is mounted.

2.2 The NFS Protocol

1. provides a set of remote proc calls for remote file operations. They support
   - searching for a file within a directory
   - reading a set of directory entries
   - manipulating links and directories
   - accessing file attributes
   - reading and writing files
2. stateless-server

   no open and close file commands

2.3 NFS Architecture

three layers

1. Unix file system interface layer

   open, read, write, close

2. Virtual File System (VFS) layer

   - separates file-system generic ops. from their inplementation by defining clear VFS interface (could support multiple implementations)
   - use vnode (virtual i-node) – id for a file that is network wide unique.

Stateless-server
$\Longrightarrow$ modified data must be committed to the server's disk before results are returned to the client.
$\Longrightarrow$ performance penalty

A single NFS write call is atomic
No concurrency control
A write system call is broken into several RPC write (each upto 8K). So two user writing to the same remote file can be intermixed.

Remote Op

- two caches (client)
  file-blocks
  file-attributes (inode-infor)

- on a file open, the kernel checks with the remote server whether to fetch or revalidate the cached attributes. The cached blocks usd if the cached attributes are upto date. The attribute cache is updated whenever new attributes arrive from the server.

- no cache coherence guarantee
  use timer to expire cache,
  3 sec for data block
  30 sec for directory

3 The Sprite File System (SFS)

for a network of workstations
U.C.-Berkeley
Goals:

1. (hide the distributed nature of a network) provide efficient data sharing and comm.

2. provide improved performance by using the large phy. mem. (available in today's workstation) for file caching.

3. to support Unix file semantics.

3.1 Naming

1. The entire filename space appears as a single hierarchy. (Fig 9.5)

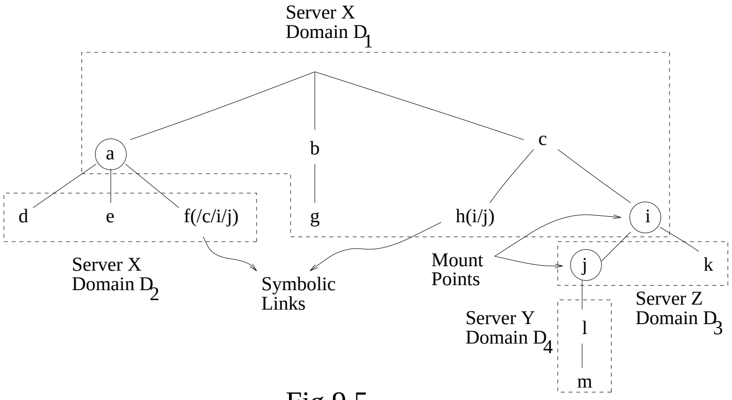


Fig 9.5

2. Servers Y - Z each store one domain. X stores two domains(D_1 and D_2)

3. nodes a, i, j are mount points.

4. Each Client has a prefix table to look up files (Table 9.1)

5. locating a file
   e.q. /a/d
   select an entry in the prefix table with longest matching prefix.
   /a
   /d and token D_2 to the server X.

6. managing prefix tables. (used as hints)

   - Initially empty
   - A client broadcast the filename when not found in its prefix table.
   - A server with match domain replies its address, its prefix token, and the prefix name corresponding to the topmost directory in the domain.
   - The client updates its prefix table.
   - The prefix tables created and used as hints.
   - If open file fails to locate a file, then update the prefix table.
   - If read and write fails due to the server failure, it is treated as a hard disk error, and an error is returned to the user.

3.2 Cache

1. The client-cache in the main-memory

2. referenced using virtual address (not phy. addr) and the block number wiithin the file
   Adv $\Longrightarrow$ client can create new blocks without having to contact the server.

3. Directories not cached at clients to avoid inconsistency.

4. a client reads a file block.
   check the client-cache
   check the local-disk (if local file)
   forward read req to the remote server
   check the server-cache
   check the server-disk

5. Sprite does not prefetch data blocks.

6. Delay writing policy
   why?

   (a) 20 30% of new data deleted within 30 secs.

   (b) 75% open for less than .5 secs

   (c) 90% open for less than 10 secs

   The traffic between servers and clients can be reduced by not updating files immediately.

7. The delayed writing policy works as follows
   every 5 secs, checks the client cache for blocks not been modified in 30 sects. Such blocks are written back to the server's cache. Additional 30 60 secs to write the server's disk.

8. Cache block replacement.
   80%, the replacement is for another file block
   20%, the replacement is for VM.
   On Avg, cache blocks not used for an hour is replaced.

9. Cache Miss
   40% of read request
   1% of write request

10. Cache Consistency
    Server initiated approach
    Concurrent-write problem avoided by disabling the cache for files open concurrently for reads and writes
    1% of network traffic due to the uncaching of shared files.

11. VM & Cache complete for main memory

4 Andrew

Predecessor of Coda
Goal : Scalable
Separate Client and Server machines
Clients file name space

- local name space
- global name space

Clusters of Clients and Servers for scalability
Cross-cluster reference relatively infrequent
Server CPU speed is system's bottleneck.
Whole file caching
Vice: dedicated servers
Virtual protocol for workstations to communicate with Vice.
Each cluster has a cluster server (designated vice)
low level id : fid.

- 96 bits long
- volume #, vnode #, 32 bits each

Volume-location database (replicated on each server)
Caching entire files.
Venus - handle file system calls at client.
Venus does not need to contact Vice on a file open to validate cached copy.
Callback policy –

- If a client updates and close a file on a server, it takes away callback on locally cached copy. When a locally cached file is open, then it is invalidated and new copy is received from the server.

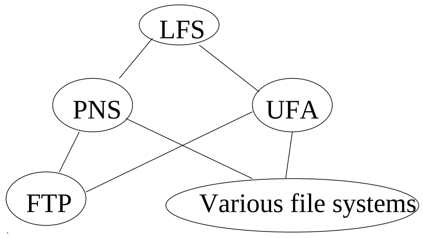
5 Coda

a descendant of Andrew
Goals :

1. scalability
2. constant data availability
3. graceful integration of the use of the file system with portable computers.
4. increase data availability by
   - the replication of files across servers protects against server failures and network failures
   - the ability of a client to operate entirely out of its cache, when no server can be contacted.

6 The X-kernel Logical File System

1. provides a uniform interface with which to access heterogeneous physical file systems.
2. provides only directory service.



3. Each user defines his private file system out of existing physical file system. (Fig 9.7)
   Lmkdir (<name>, <physical file directory>)

4. PNS : private name space protocol
   UFA : uniform file access protocol
   LFS : logical file system

5. UFA translates file operations read, write, ... into appr. low-level commands for phy. file system.

6. Handles caching
   If a file sys. is not supported, UFA invokes FTP to retrieve the entire fiile.

7. PNS superimpose a logical hierarchy over a collection of existing file hierarchies. (Fig. 9.7)

8. A directory entry (3 parts)
   (a) logical file name
   (b) phy. directory, phy. file sys. type, server.
   (c) pt. to a private directory.

$$\underbrace{\text{/text/doc/journal}}_{\text{PNS}}\underbrace{\text{/paper/1}}_{\text{NFS}}$$

9. the pathnames in logical file system takes precedence over the pathnames of the physical file system.