Distributed File Systems

Chapter 10

Distributed File System

a) A **distributed file system** is a file system that resides on different machines, but offers an integrated view of data stored on remote disks.

b) Examples of distributed file systems

  a) NFS

  b) AFS

  c) Coda

  d) Plan9

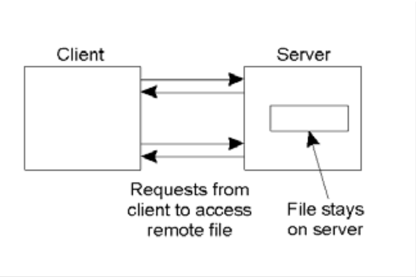
  e) xFS

Network File System (NFS)

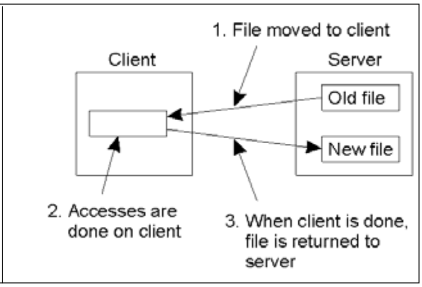
- Developed originally at Sun Microsystems for UNIX workstations.
- It is a model to integrate different file systems.
- Based on the idea that each file server provides a standardized view of its local file system.
- NFS runs on heterogeneous groups of computers.

NFS Architecture (1)

- NFS uses a **remote access model**: clients are unaware of file locations.
- Servers export a set of operations on files.

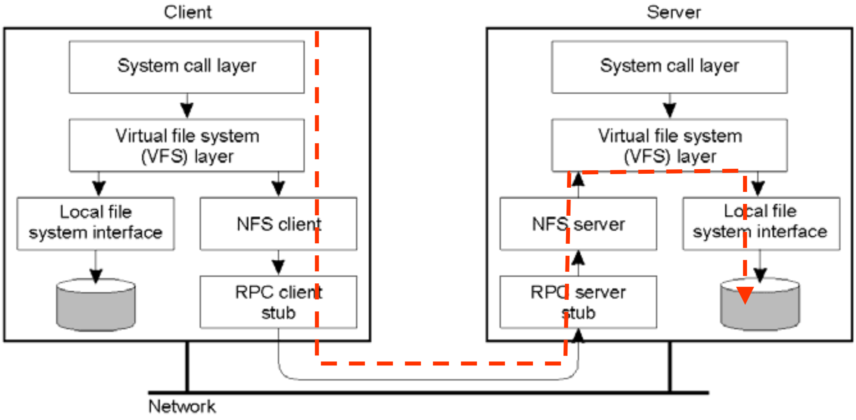


The remote access model.


1. File moved to client
Client
Server
Old file
New file
2. Accesses are done on client
3. When client is done, file is returned to server

The upload/download model.

NFS Architecture (2)

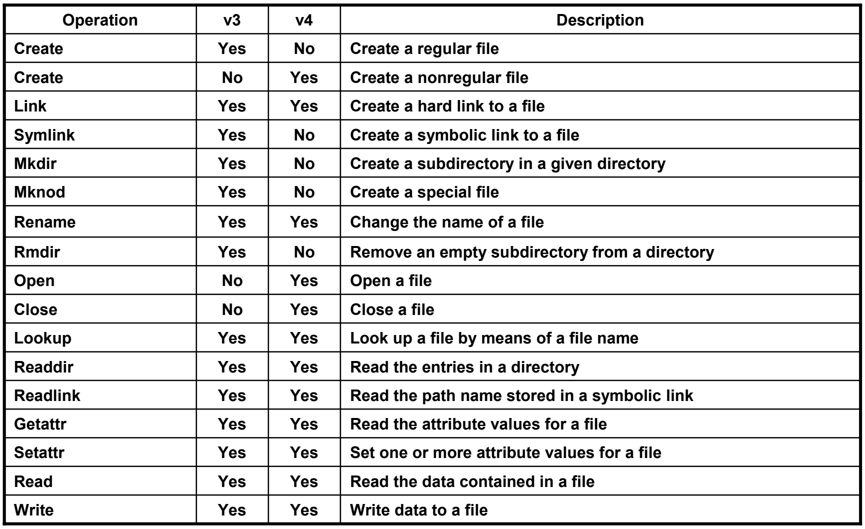

Client
System call layer
Virtual file system (VFS) layer
Local file system interface
NFS client
RPC client stub
Server
System call layer
Virtual file system (VFS) layer
NFS server
Local file system interface
RPC server stub
Network

The basic NFS architecture for UNIX systems.

NFS Architecture (3)

- NFS is independent from local file system organization.
- It integrates file systems used in UNIX, Linux, Windows, and other operating systems.
- The model offered is similar to UNIX-like file systems based on files as sequences of bytes.

File System Model



Operation	v3	v4	Description
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file
Link	Yes	Yes	Create a hard link to a file
Symlink	Yes	No	Create a symbolic link to a file
Mkdir	Yes	No	Create a subdirectory in a given directory
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Rmdir	Yes	No	Remove an empty subdirectory from a directory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a file name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name stored in a symbolic link
Getattr	Yes	Yes	Read the attribute values for a file
Setattr	Yes	Yes	Set one or more attribute values for a file
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

An incomplete list of file system operations supported by NFS.

Communication (1)

a) In NFS all communications between servers and clients are implemented using Remote Procedure Call (RPC).

b) The used protocol is the Open Network Computing RPC.

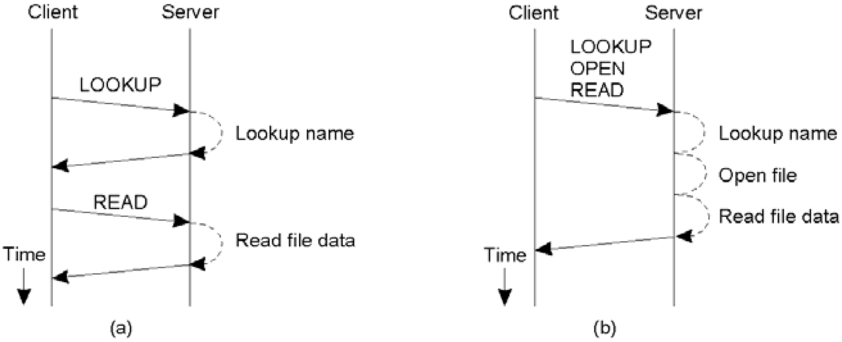
c) Before version 4, NFS used **stateless** servers.

d) The clients were in charge to maintain the status of current operations on a remote file system.

Communication (2)

- In version 4, NFS introduced **compound operations** to improve the reduce the number of RPC calls and improve communication performance.
- This is appropriate for wide-area file systems.
- Compound operations are not handled as transactions.
- If one operation in a compound procedure fails successive operations are not executed.

Communication (3)



(a) Reading data from a file in NFS version 3.

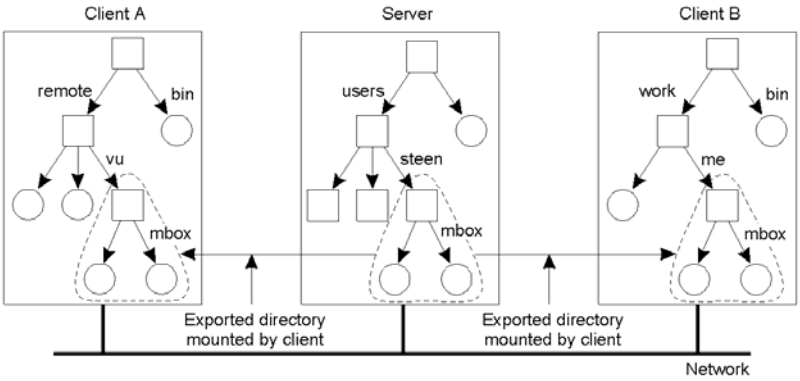
(b) Reading data using a compound procedure in version 4.

Communication (4)

- In version 4, NFS servers maintain the status of some operations.
- This model was introduced to handle with wide-area network operations such as
  - file locking
  - cache consistency protocols
  - callback procedures.

Naming (1)

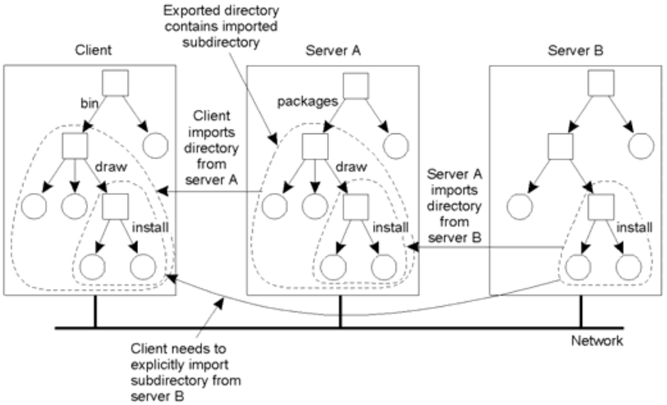
File sharing is based on **mounting** operations.


Client A
Server
Client B
remote
bin
vu
mbox
users
steen
mbox
work
bin
me
mbox
Exported directory mounted by client
Exported directory mounted by client
Network

Mounting (part of) a remote file system in NFS.

Naming (2)

An NFS server can mount directories exported from other servers, but these cannot be exported to clients.

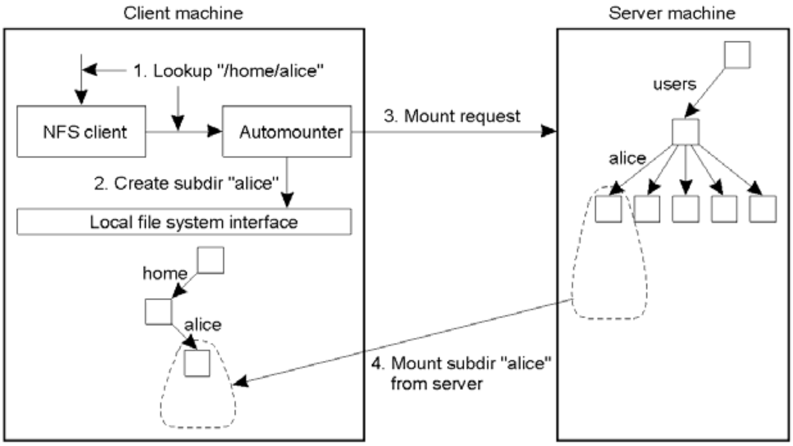

Exported directory contains imported subdirectory
Client
Server A
Server B
bin
packages
draw
draw
install
install
install
Client imports directory from server A
Server A imports directory from server B
Network
Client needs to explicitly import subdirectory from server B

Mounting nested directories from multiple servers in NFS.

Automounting (1)

- When a file system should be mounted on a client ?
- An automatic procedure is implemented by an **automounter** for NFS that
  - mount home directories of users when they log into the client and
  - mount other file system on demand (when files are accessed).

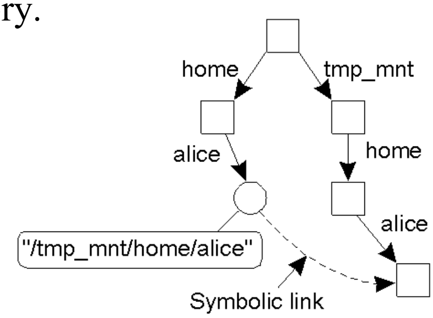
Automounting (2)



A simple automounter for NFS.

Automounting (3)

- To avoid to call the automounter whenever a file is read, directories can be mounted on a special sub-directory and using a symbolic link to each mounted directory.



Using symbolic links with automounting.

File Attributes (1)

- NFS file attributes are divided between two groups: 12 **mandatory** (supported by every implementation) and 43 **recommended** attributes.

Attribute	Description
TYPE	The type of the file (regular, directory, symbolic link)
SIZE	The length of the file in bytes
CHANGE	Indicator for a client to see if and/or when the file has changed
FSID	Server-unique identifier of the file's file system

Some general mandatory file attributes in NFS.

File Attributes (2)

Attribute	Description
ACL	an access control list associated with the file
FILEHANDLE	The server-provided file handle of this file
FILEID	A file-system unique identifier for this file
FS_LOCATIONS	Locations in the network where this file system may be found
OWNER	The character-string name of the file's owner
TIME_ACCESS	Time when the file data were last accessed
TIME_MODIFY	Time when the file data were last modified
TIME_CREATE	Time when the file was created

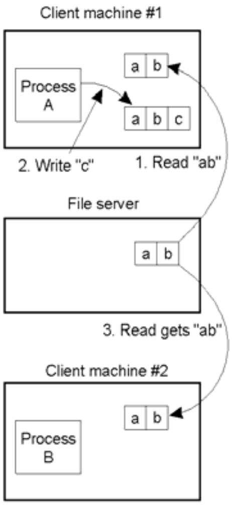
Some general recommended file attributes.

Semantics of File Sharing (1)

- According to the **UNIX semantics** in a sequential system that allows to share files
  - a read after a write, returns the value just written
  - after two successive writes a read operation returns the value stored by the last write.
- In a distributed system, UNIX semantics can be assured if there is only one file server and clients do not cache files.

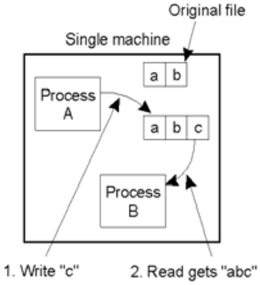
Semantics of File Sharing (2)

In a distributed system with caching, obsolete values may be returned.



(b)

On a single machine, when a *read* follows a *write*, the value returned by the *read* is the value just written.



(a)

Semantics of File Sharing (3)

- Although NFS in theory uses the remote access model, most implementation use local caches, so they in practice use the upload/download model.
- NSF implements the **session semantics**:

  changes to an open file are initially visible only to the process that modified the file . When the file is closed all the changes are visible to other processes (or machines).
- What happens when two processes caches and modify a file?

Semantics of File Sharing (4)

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transaction	All changes occur atomically

Four ways of dealing with the shared files in a distributed system.

File Locking in NFS (1)

- NFS version 4 use a file locking method.

File Locking in NFS (1)

- NFS version 4 use a file locking method.
- Read locks are not mutually exclusive.
- Write lock is exclusive.

Operation	Description
Lock	Creates a lock for a range of bytes
Lockt	Test whether a conflicting lock has been granted
Locku	Remove a lock from a range of bytes
Renew	Renew the leas on a specified lock

NFS version 4 operations related to file locking.

File Locking in NFS (2)

- NFS implements an implicit way to lock a file: **share reservation**

Current file denial state

Request access

	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

(a)

Requested file denial state

Current access state

	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

(b)

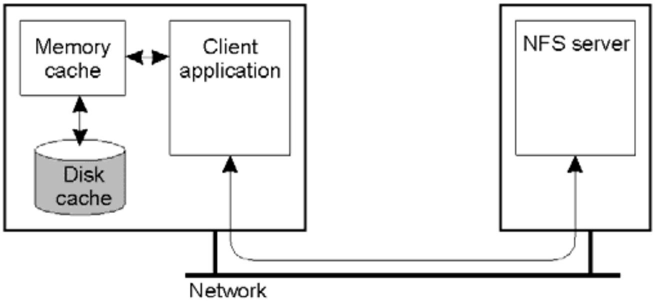
The result of an *open* operation on an already opened by another client with share reservations in NFS.

(a) When the client requests shared access given the current denial state.

(b) When the client requests a denial state given the current file access state.

NFS Client Caching (1)

- NFS version 4 provides Client-side caching including a Memory cache and a Disk cache.
- File data, attributes, handles, and directories can be cached.

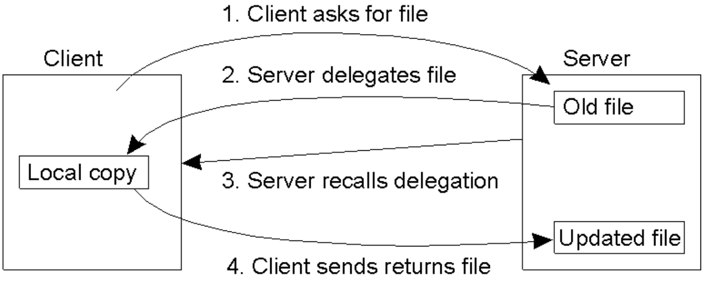


NFS Client Caching (2)

- Caching of file data uses the session semantics: modification of cached data must be flushed to the server when a client closes the file.
- Data can be retained in the cache, but if the file will be re-opened they must be revalidated.
- NFS uses **open delegation** to delegate some rights to a client that opened a file.
- The client can take some decisions without asking the server. Some other decisions remain to the server.

NFS Client Caching (3)

- An NFS may need to recall a delegation when another client on a different machine asks for access rights to a file.
- The **callback** mechanism is used to recall file delegation.



NFS Client Caching (4)

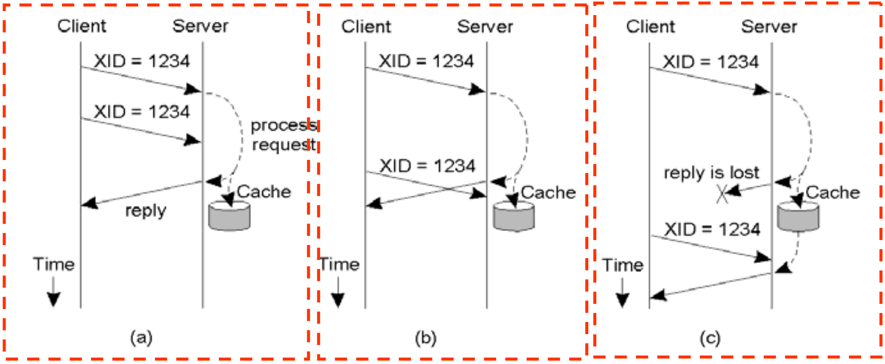
- Attribute values, file handles, and directories can be cached, but modifications to those values must be sent to the server.
- Cache entries are automatically invalidated after a certain amount of time. This oblige clients to revalidate them before to use them again.
- NFS v4 provides a support for file system **replication** through a list of locations of a file system.

NFS Fault Tolerance

- As NFS v4 provides stateful servers (e.g., file locking, open delegation), fault tolerance and recovery mechanisms need to be designed to handle with RPC failures.
- RPC may use TCP or UDP protocols.
- RPC may incurs in duplicate requests when an RPC reply is lost; so the server can carry out the request more than one time.

Duplicate-Request Cache

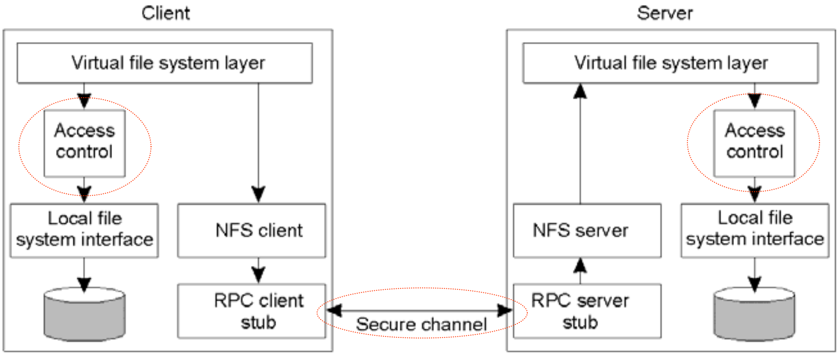
Each RPC request from a client carries a unique transaction id (**XID**) and it is cached by the server with the reply.



Three situations for handling retransmissions.
(a) The request is still in progress
(b) The reply has just been returned
(c) The reply has been some time ago, but was lost.

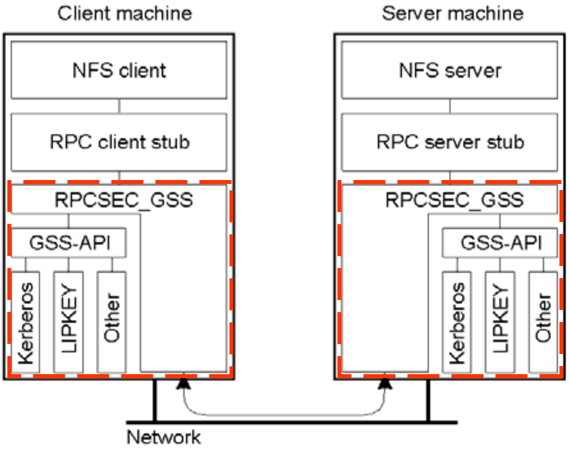
NFS Security

Security in NFS is mainly based on **secure channels** and **file access control**.


Client
Virtual file system layer
Access control
Local file system interface
NFS client
RPC client stub
Secure channel
Server
Virtual file system layer
Access control
NFS server
Local file system interface
RPC server stub

The NFS security architecture.

Secure RPCs



Secure RPC in NFS version 4 is based on RPCSRC_GSS.

Access Control

Values of the ACL attribute

Operation	Description
Read_data	Permission to read the data contained in a file
Write_data	Permission to to modify a file's data
Append_data	Permission to to append data to a file
Execute	Permission to to execute a file
List_directory	Permission to to list the contents of a directory
Add_file	Permission to to add a new file t5o a directory
Add_subdirectory	Permission to to create a subdirectory to a directory
Delete	Permission to to delete a file
Delete_child	Permission to to delete a file or directory within a directory
Read_acl	Permission to to read the ACL
Write_acl	Permission to to write the ACL
Read_attributes	The ability to read the other basic attributes of a file
Write_attributes	Permission to to change the other basic attributes of a file
Read_named_attrs	Permission to to read the named attributes of a file
Write_named_attrs	Permission to to write the named attributes of a file
Write_owner	Permission to to change the owner
Synchronize	Permission to to access a file locally at the server with synchronous reads and writes

The classification of operations recognized by NFS with respect to access control.

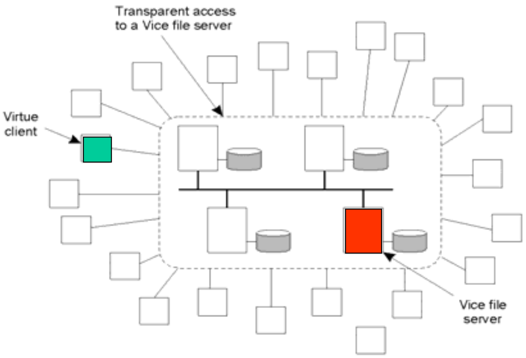
The NFS User Types

Type of user	Description
Owner	The owner of a file
Group	The group of users associated with a file
Everyone	Any user of a process
Interactive	Any process accessing the file from an interactive terminal
Network	Any process accessing the file via the network
Dialup	Any process accessing the file through a dialup connection to the server
Batch	Any process accessing the file as part of a batch job
Anonymous	Anyone accessing the file without authentication
Authenticated	Any authenticated user of a process
Service	Any system-defined service process

The various kinds of users and processes distinguished by NFS with respect to access control.

Overview of Coda (1)

- Coda is based on the Andrew File System (AFS).
- Goals: **naming and location transparency** and **high availability**.

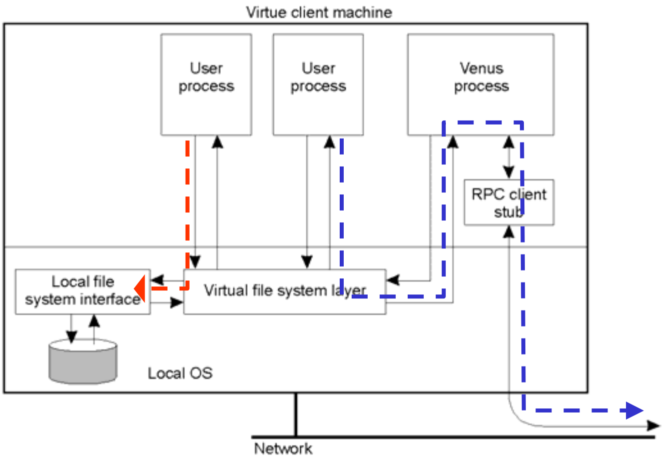


The overall organization of AFS.

Overview of Coda (2)

- In each Virtue client is running a Venus process that plays the same role of an NFS client.
- Venus role is also to allows the client to continue to work even if the file server access is not possible.
- Communication is based on **reliable RPC**.

Overview of Coda (3)



The internal organization of a Virtue workstation.

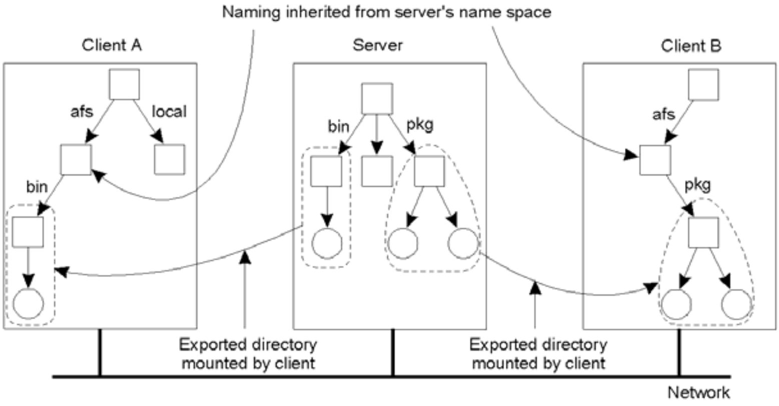
Overview of Coda (4)

- Coda implements a UNIX-like file system with similar operations of NFS.
- Coda provides a global shared name space maintained by Vice servers
- Clients access the global name space through a special subdirectory (*/afs*).
- When accessed, a part of the shared name space is mounted locally.

Naming in Coda (1)

- Namining in Coda is similar to that of UNIX.
- File are grouped in **volumes** - disk partitions that correspond to file systems associated to a user and stored in a Vice server.
- Differently form NFS, in Coda shared file have the same name.
- Coda uses Logical volumes and Replicated Volume Identifiers (RVI).

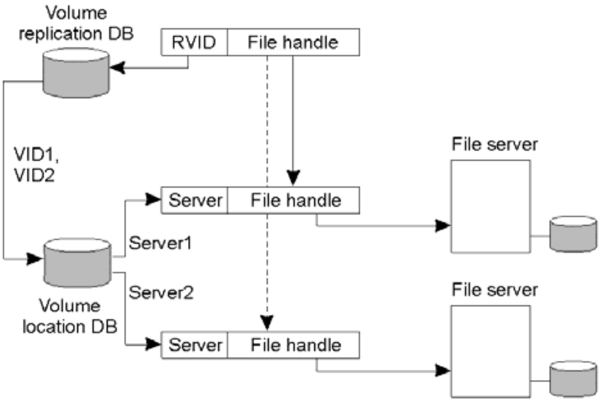
Naming in Coda (2)


Naming inherited from server's name space
Client A
Server
Client B
afs
local
bin
bin
pkg
afs
pkg
Exported directory mounted by client
Exported directory mounted by client
Network

Clients in Coda have access to a single shared name space.

File Identifiers

File ids are composed of two parts: **RVID** + **vnode**



The implementation and resolution of a Coda file identifier.

Transactional Semantics

Coda implements a form of weak transactional semantics by interpreting a session as a transaction.

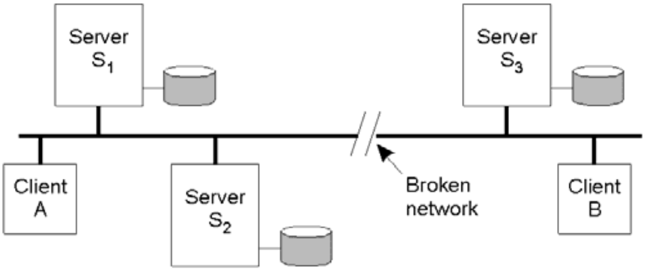
Different types of sessions are defined and different system calls are associated to a session type.

File-associated data	Read?	Modified?
File identifier	Yes	No
Access rights	Yes	No
Last modification time	Yes	Yes
File length	Yes	Yes
File contents	Yes	Yes

The metadata read and modified for a *store* session type in Coda.

Server Replication

- Coda allows replicated file servers called Volume Storage Group (**VSG**).
- For each client an Accessible VSG is provided and a replicated-write protocol is used for consistency.



Two clients with different AVSG for the same replicated file.

Access Control

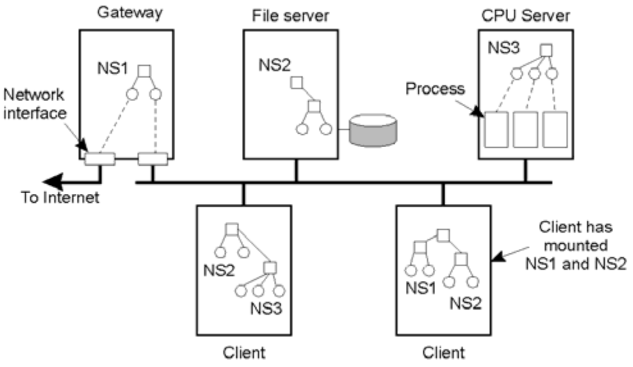
- Access control lists are associated with directories not with files.

Operation	Description
Read	Read any file in the directory
Write	Modify any file in the directory
Lookup	Look up the status of any file
Insert	Add a new file to the directory
Delete	Delete an existing file
Administer	Modify the ACL of the directory

Classification of file and directory operations recognized by Coda with respect to access control.

Plan 9: Resources Unified to Files

- All resources are accessed using a file-like syntax on a pool of servers.



General organization of Plan 9

Communication

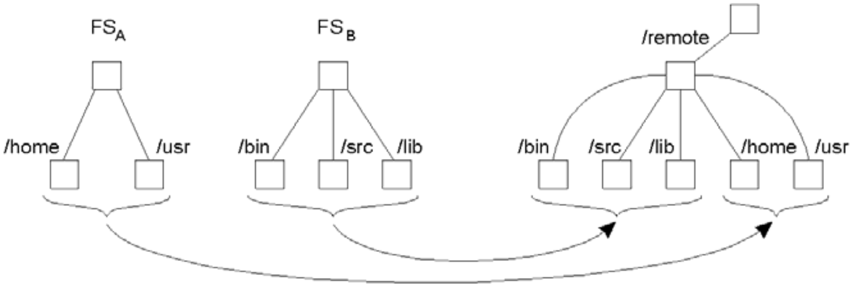
- For communications Plan 9 uses the 9P protocol and network interfaces are represented as directories.

File	*Description*
ctl	Used to write protocol-specific control commands
data	Used to read and write data
listen	Used to accept incoming connection setup requests
local	Provides information on the caller's side of the connection
remote	Provides information on the other side of the connection
status	Provides diagnostic information on the current status of the connection

Files associated with a single TCP connection in Plan 9.

Naming

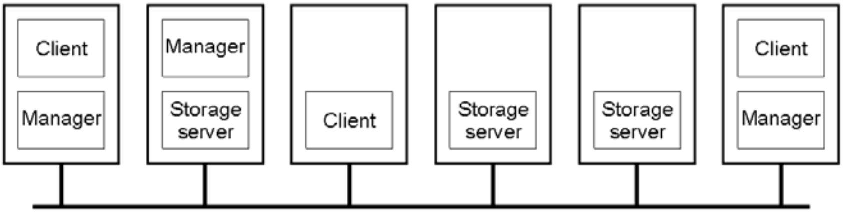
- A client can mount multiple name spaces at the same mount point composing a **union directory**.
- The mounting order is maintained in file search.



A union directory in Plan 9.

Overview of xFS.

- The xFS file system is based on a **serverless** model.
- The entire file system is distributed across machines including clients.
- Each machine can run a storage server, a metadata server and a client process.



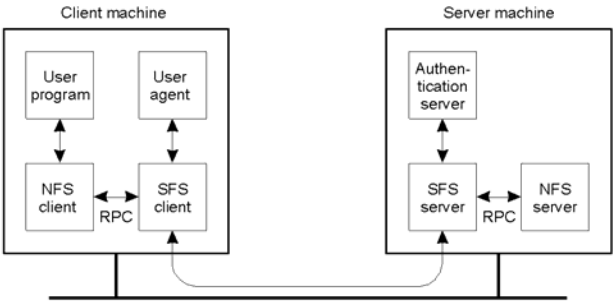
A typical distribution of xFS processes across multiple machines.

Communication in xFS

- RPC was substituted with **active messages** in xFS.
- RPC performance was not the best and fully decentralization is hard to manage with RPC.
- In an **active message**, when a message arrives, an handler is automatically invoked for execution.

Overview of SFS

- The Secure File System uses keys for file system security.
- Clients cannot access a file without having a secret key.



The organization of SFS.

Summary

Issue	NFS	Coda	Plan 9	xFS	SFS
Design goals	Access transparency	High availability	Uniformity	Serverless system	Scalable security
Access model	Remote	Up/Download	Remote	Log-based	Remote
Communication	RPC	RPC	Special	Active msgs	RPC
Client process	Thin/Fat	Fat	Thin	Fat	Medium
Server groups	No	Yes	No	Yes	No
Mount granularity	Directory	File system	File system	File system	Directory
Name space	Per client	Global	Per process	Global	Global
File ID scope	File server	Global	Server	Global	File system
Sharing sem.	Session	Transactional	UNIX	UNIX	N/S
Cache consist.	write-back	write-back	write-through	write-back	write-back
Replication	Minimal	ROWA	None	Striping	None
Fault tolerance	Reliable comm.	Replication and caching	Reliable comm.	Striping	Reliable comm.
Recovery	Client-based	Reintegration	N/S	Checkpoint & write logs	N/S
Secure channels	Existing mechanisms	Needham-Schroeder	Needham-Schroeder	No pathnames	Self-cert.
Access control	Many operations	Directory operations	UNIX based	UNIX based	NFS BASED

A comparison between NFS, Coda, Plan 9, xFS. N/S indicates that nothing has been specified.