Distributed Systems
Principles and Paradigms

Chapter 10
(version 27th November 2001)

Maarten van Steen

Vrije Universiteit Amsterdam, Faculty of Science
Dept. Mathematics and Computer Science
Room R4.20. Tel: (020) 444 7784
E-mail:steen@cs.vu.nl, URL: www.cs.vu.nl/∼steen/

Distributed File Systems

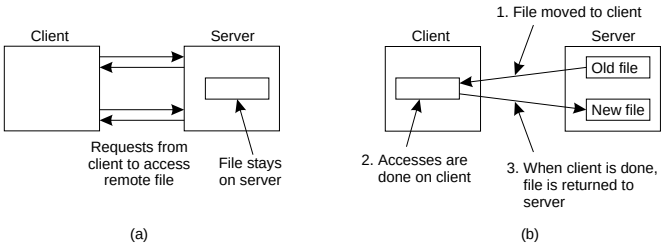
- Sun NFS
- Coda

Sun NFS

Sun Network File System: Now version 3, version 4 is coming up.

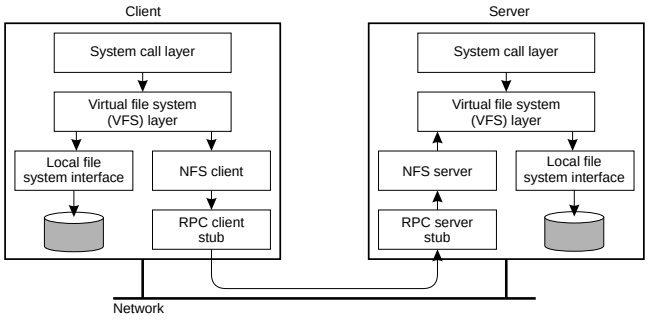
Basic model: Remote file service: try to make a file system transparently available to remote clients.

Follows **remote access model** (a)
instead of **upload/download model** (b):



NFS Architecture

NFS is implemented using the **Virtual File System** abstraction, which is now used for lots of different operating systems:



Essence: VFS provides standard file system interface, and allows to hide difference between accessing local or remote file system.

Question: Is NFS actually a file system?

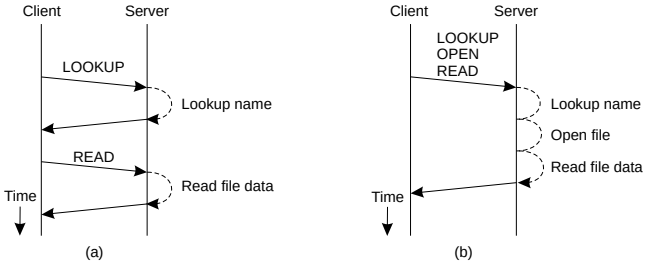
NFS File Operations

Oper.	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
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Remove	Yes	Yes	Remove a file from a file system
Rmdir	Yes	No	Remove an empty subdirectory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name in a symbolic link
Getattr	Yes	Yes	Get the attribute values for a file
Setattr	Yes	Yes	Set one or more file-attribute values
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

Question: Anything unusual between v3 and v4?

Communication in NFS

Essence: All communication is based on the (best-effort) Open Network Computing RPC (ONC RPC). Version 4 now also supports **compound procedures**:



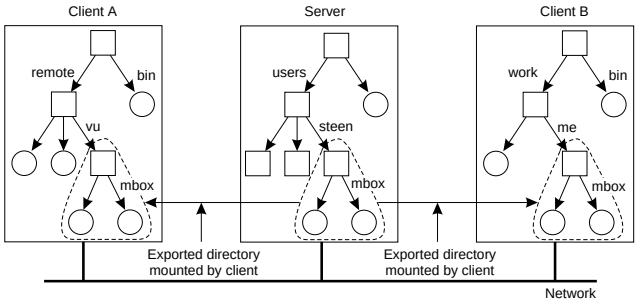
(a) Normal RPC

(b) Compound RPC: first failure breaks execution of rest of the RPC

Question: What's the use of compound RPCs?

Naming in NFS (1/2)

Essence: NFS provides support for mounting remote file systems (and even directories) into a client's local name space:

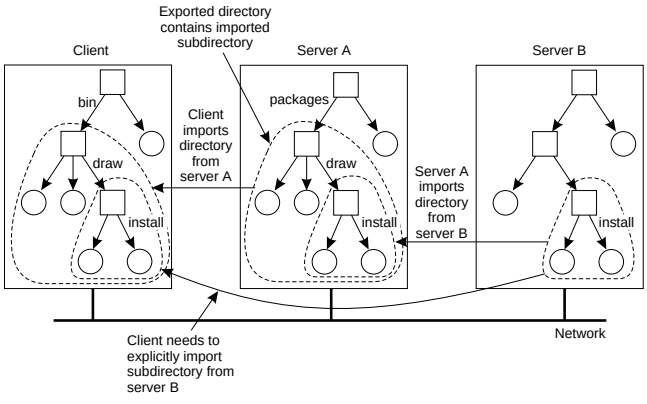


Watch it: Different clients may have different local name spaces. This may make file sharing extremely difficult (**Why?**).

Question: What are the solutions to this problem?

Naming in NFS (2/2)

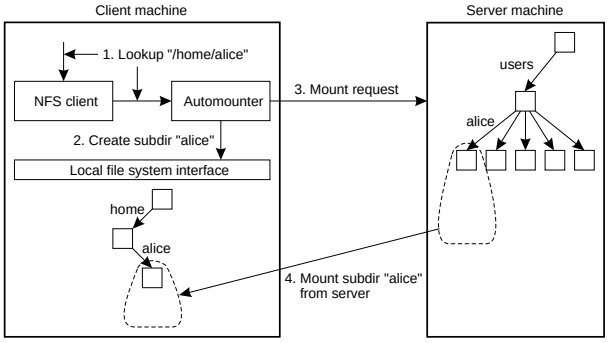
Note: A server cannot export an imported directory. The client must mount the server-imported directory:



Automounting in NFS

Problem: To share files, we partly standardize local name spaces and mount shared directories. Mounting very large directories (e.g., all subdirectories in /home/users) takes a lot of time (**Why?**).

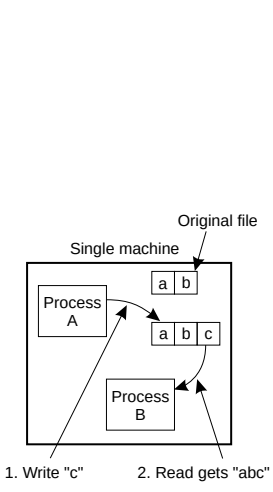
Solution: Mount on demand — **automounting**



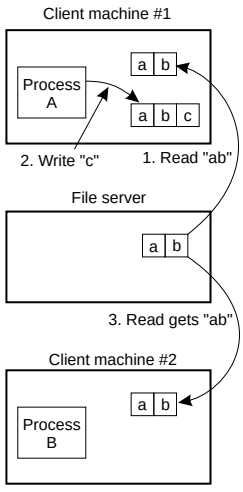
Question: What's the main drawback of having the automounter in the loop?

File Sharing Semantics (1/2)

Problem: When dealing with distributed file systems, we need to take into account the ordering of concurrent read/write operations, and expected semantics (= consistency).



(a)



(b)

File Sharing Semantics (2/2)

UNIX semantics: a `read` operation returns the effect of the last `write` operation $\Rightarrow$ can only be implemented for remote access models in which there is only a single copy of the file

Transaction semantics: the file system supports transactions on a *single* file $\Rightarrow$ issue is how to allow concurrent access to a physically distributed file

Session semantics: the effects of `read` and `write` operations are seen only to the client that has opened (a local copy) of the file $\Rightarrow$ what happens when a file is closed (only one client may actually win)

File Locking in NFS

Observation: It could have been simple, but it isn't. NFS supports an explicit locking protocol (stateful), but also an implicit **share reservation** approach:

		Current denial state			
		None	**Read**	**Write**	**Both**
Request access	**Read**	OK	Fail	OK	Fail
	Write	OK	OK	Fail	Fail
	Both	OK	Fail	Fail	Fail

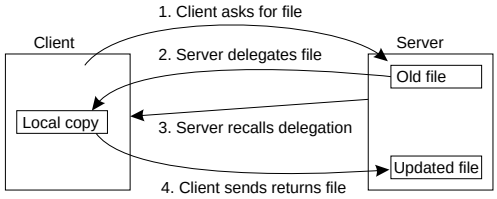
		Requested denial state			
		None	**Read**	**Write**	**Both**
Current access state	**Read**	OK	Fail	OK	Fail
	Write	OK	OK	Fail	Fail
	Both	OK	Fail	Fail	Fail

Question: What's the use of these share reservations?

Caching & Replication

Essence: Clients are on their own.

Open delegation: Server will explicitly permit a client machine to handle local operations from other clients on that machine. Good for performance. Does require that the server can take over when necessary:



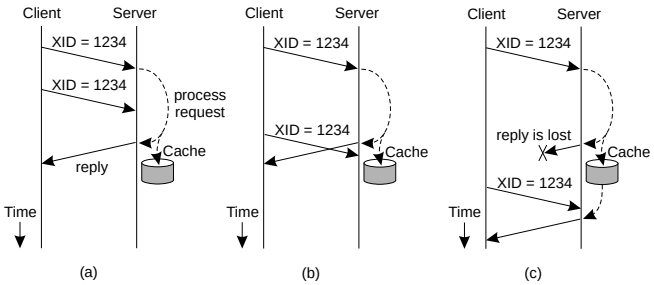
Question: Would this scheme fit into v3

Question: What kind of file access model are we dealing with?

Fault Tolerance

Important: Until v4, fault tolerance was easy due to the stateless servers. Now, problems come from the use of an unreliable RPC mechanism, but also stateful servers that have delegated matters to clients.

RPC: Cannot detect duplicates. Solution: use a duplicate-request cache:



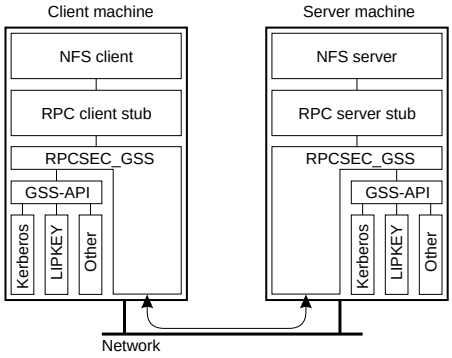
Locking/Open delegation: Essentially, recovered server offers clients a grace period to **reclaim** locks. When period is over, the server starts its normal local manager function again.

Security

Essence: Set up a secure RPC channel between client and server:

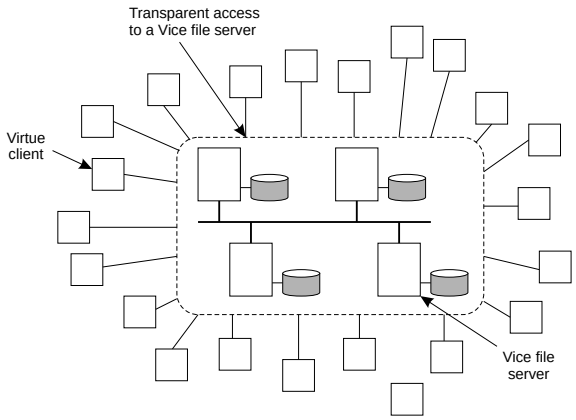
Secure NFS: Use Diffie-Hellman key exchange to set up a secure channel. However, it uses only 192-bit keys, which have shown to be easy to break.

RPCSEC_GSS: A standard interface that allows integration with existing security services:

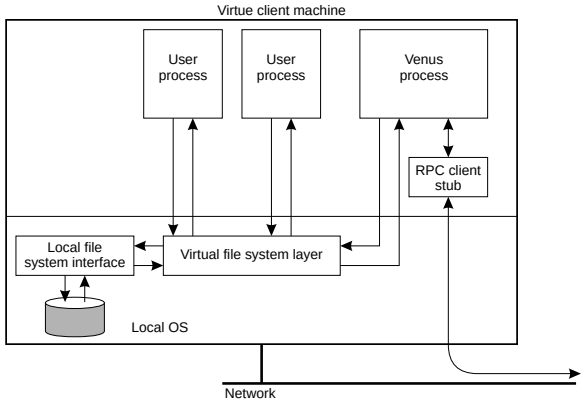


Coda File System

- Developed in the 90s as a descendant of the Andrew File System (CMU)
- Now shipped with Linux distributions (after 10 years!)
- Emphasis: support for mobile computing, in particular disconnected operation.



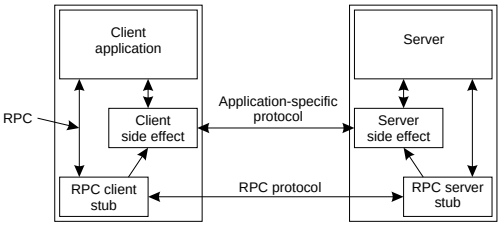
Coda Architecture



Note: The core of the client machine is the Venus process (yes, it's a nasty beast). Note that most stuff is at user level.

Communication in Coda (1/2)

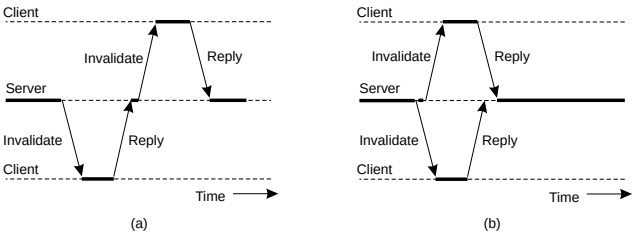
Essence: All client-server communication (and server-server communication) is handled by means of a reliable RPC subsystem. Coda RPC supports **side effects**:



Note: side effects allows for separate protocol to handle, e.g., multimedia streams.

Communication in Coda (2/2)

Issue: Coda servers allow clients to cache whole files. Modifications by other clients are notified through invalidation messages $\Rightarrow$ there is a need for **multicast RPC**:



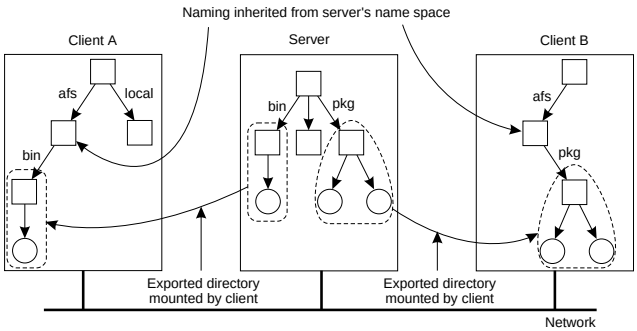
(a) Sequential RPCs

(b) Multicast RPCs

Question: Why do multi RPCs *really* help?

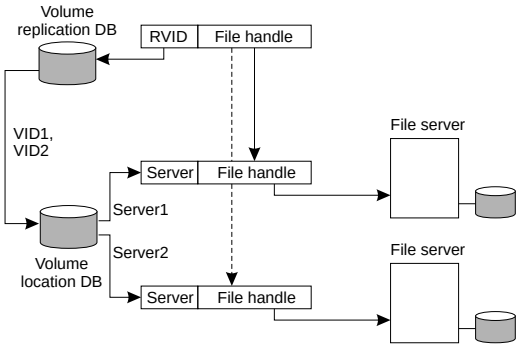
Naming in Coda

Essence: Similar remote mounting mechanism as in NFS, except that there is a shared name space between all clients:



File Handles in Coda

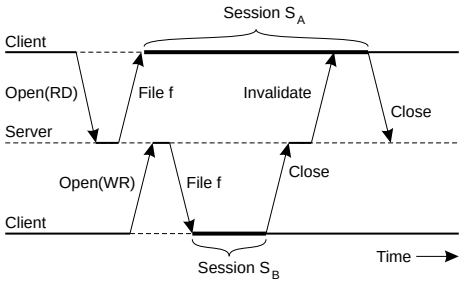
Background: Coda assumes that files may be replicated between servers. Issue becomes to track a file in a location-transparent way:



- Files are contained in a **volume** (cf. to UNIX file system on disk)
- Volumes have a Replicated Volume Identifier (RVID)
- Volumes may be replicated; physical volume has a VID

File Sharing Semantics in Coda

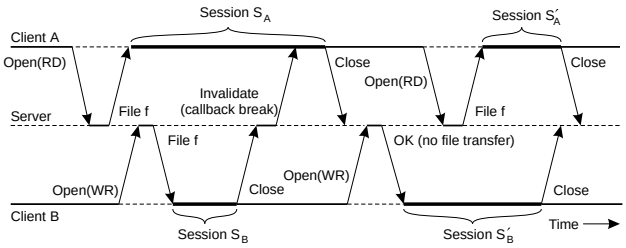
Essence: Coda assumes transactional semantics, but without the full-fledged capabilities of real transactions.



Note: Transactional issues reappear in the form of "this ordering could have taken place."

Caching in Coda

Essence: Combined with the transactional semantics, we obtain flexibility when it comes to letting clients operate on local copies:



Note: A writer can continue to work on its local copy; a reader will have to get a fresh copy on the next open.

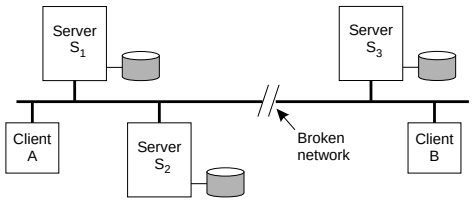
Question: Would it be OK if the reader continued to use its own local copy?

Server Replication in Coda (1/2)

Essence: Coda uses ROWA for server replication:

- Files are grouped into volumes (cf. traditional UNIX file system)
- Collection of servers replicating the same volume form that volume's **Volume Storage Group**)
- Writes are propagated to a file's VSG
- Reads are done from one server in a file's VSG

Problem: what to do when the VSG partitions and partition is later healed?



Server Replication in Coda (2/2)

Solution: Detect inconsistencies using version vectors:

- $CVV_i(f)[j] = k$ means that server S_i knows that server S_j has seen version k of file f.
- When a client reads file f from server S_i, it receives $CVV_i(f)$.
- Updates are multicast to all reachable servers (client's **accessible VSG**), which increment their $CVV_i(f)[i]$.
- When the partition is restored, comparison of version vectors will allow detection of conflicts and possible reconciliation.

Note: the client informs a server about the servers in the AVSG where the update has also taken place.

Fault Tolerance

Note: Coda achieves high availability through client-side caching and server replication

Disconnected operation: When a client is no longer connected to one of the servers, it may continue with the copies of files that it has cached. Requires that the cache is properly filled (**hoarding**).

- Compute a priority for each file
- Bring the user's cache into equilibrium (**hoard walk**):
  - There is no uncached file with higher priority than a cached file
  - The cache is full or no uncached file has nonzero priority
  - Each cached file is a copy of a file maintained by the client's AVSG

Note: Disconnected operation works best when there is hardly any write-sharing .

Security

Essence: All communication is based on a secure RPC mechanism that uses secret keys. When logging into the system, a client receives:

- A clear token CT from an AS (containing a generated shared secret key K_S). CT has time-limited validity.
- A secret token $ST = K_{vice}([CT]^*_{K_{vice}})$, which is an encrypted and cryptographically sealed version of CT.