FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Distributed Systems
Distributed File Systems

Prof. Dr. Oliver Hahm
Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

Data Storage Systems

	File systems	Database systems	Object management systems
Content	universal	mass data of a few structural types	focus on relationships
Stored information	passive	passive	active
Semantics defining code	external	external	internal (via types)
access	by name, simple navigation	complex associative search functions	complex search and navigation functions

Backup and Disaster Recovery

Backup

Describes the process of duplicating data to a remote location in order to provide an alternative source for the data in case the primary source becomes unavailable.

Disaster Recovery

Describes the entire process to safeguard against various types of problems and restore it in the case of an failure. Backups are an essential part of disaster recovery.

Various ways to manage remote data for disaster recovery

- Backup via file transfer (e.g., `rsync`)
- Synchronization via a cloud backend (e.g., Nextcloud or Dropbox)
- Use of a version control system (e.g., Subversion or git)

Distributed file systems themselves do not provide a backup per se.

Models of File Systems

Files as a classical abstraction in operating systems

Historical development considered in the following

1 computer, 1 user, 1 process

- Problems to solve
  - Structure of the file system
  - Naming
  - Programming interface
  - Mapping to physical memory
  - Integrity
- Examples
  - PC-DOS
  - classic MacOS

Models of File Systems (2)

1 computer, 1 user, multiple processes

- Additional problems
  - **Concurrency control**
- Examples
  - OS/2

1 computer, multiple users, multiple processes

- Additional problems
  - **Security and access control**
- Examples
  - UNIX

Distributed File Systems

multiple computers, multiple users, multiple processes

- Considered in this lecture
- Additional problems
  - **Distributedness**
    - Visible overall structure
    - Access model
    - Location
    - Replication
    - Availability
    - ...
  - No access to shared block memory of nodes → **shared nothing**
  - Sharing common hard disks between nodes will be considered at the end of this lecture (→ Storage Area Networks (SAN))
- **Client/Server model**
  - Dedicated file server
- **Peer-to-Peer model**
  - Everyone can provide files

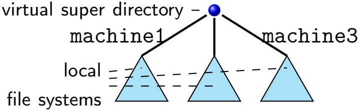
Historical Predecessors

Complete separation

- Only local access
- File transfer between isolated file systems (download/upload model)
- Example: UNIX uucp, ftp, rcp, scp

Early distributed file systems (adjunct file systems)

- Access to remote files
- Explicitly addressing the file's location as part of its name
- Example: Newcastle Connection



`/machine1/<localpath>`

`machine2!<localpath>`

`/../machine3/<localpath>`

Distributed file system

Definition

A distributed file system provides a unified file system to the users on all hosts of a network.

Which types of transparency are possible?

Distributed file system

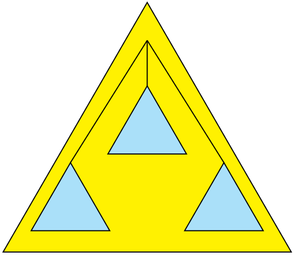
FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Definition

A distributed file system provides a unified file system to the users on all hosts of a network.

Possibly types of transparency:

- Location transparency
- Access transparency
- Replication transparency
- ...



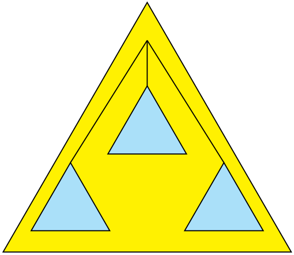
Distributed file system

Definition

A distributed file system provides a unified file system to the users on all hosts of a network.

Possibly types of transparency:

- Location transparency
  - The file name does not contain any location information
- Access transparency
- Replication transparency
- ...



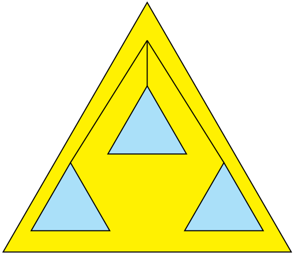
Distributed file system

Definition

A distributed file system provides a unified file system to the users on all hosts of a network.

Possibly types of transparency:

- Location transparency
  - The file name does not contain any location information
- Access transparency
  - Common API for local and remote files
- Replication transparency
- . . .



Typical Design Goals

- High degree of **transparency**
  - → previous slide
- **Performance**
  - Comparable to local access
- High **availability** and **failure tolerance**
- **Security**
- **Scalability**
- Support for **mobile nodes** with temporary disconnectivity
- Support for shared disk and shared nothing nodes
- Cloud connection

Exemplary Solutions

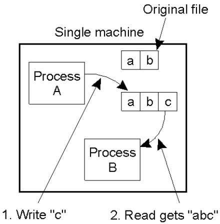
- **Network File System (NFS)** — since 1985
- **Andrew File System (AFS) + Coda** — since 1985
- **Common Internet File System (CIFS) + Server Message Block (SMB)**
- GlusterFS (Gluster Inc. → Red Hat 2011)
- IBM General Parallel File System (GPFS) (ursprünglich Cluster File System, weiterentwickelt)
- Google File System (GFS)
- Apache Hadoop
- ...

Agenda

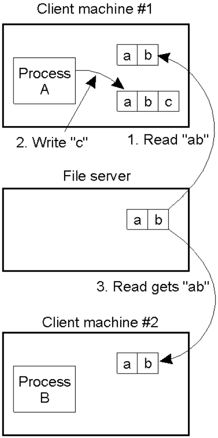
Agenda

Access Consistency Problem

- (a): Modifications are immediately visible for everyone
- (b): Visible values may be outdated



(a)



(b)

Fig. from Tanenbaum/Steen

Semantics

Strict Consistency

- Modifications are immediately visible for everyone
- Example: local UNIX

Session Semantics

- Updates the file on closing
- Allows local cache as long as the file is opened
- Example: Andrew File System

Read-Only Files

- Modifications are not possible
- Common use and replicate are significantly simplified

Transaction Semantics

- Modifications on a set of files take place in an atomic operation

Stateless and Stateful Servers

Stateless

→ Server has no memory

Advantage of stateless servers

- Recovery can be easily implemented
- No problems with client crashes
- Opening and closing of files is unnecessary
- Number of *opened* files unlimited

Advantages of stateful servers

- Shorter messages
- Higher performance
- Read-ahead possible
- Idempotence of operations easier to implement
- File locks are possible

Replication

Goals

- Availability of data
- Load balancing
- Transparency from the user perspective

Common algorithms

- Not discussed in details as part of this lecture
- Primary Copy Update
  - Primary is a dedicated copy
  - Modifications only on primary which takes care of the copies
  - Hence, primary sees all updates
  - Reading of any arbitrary copy (performance)
- Voting (Gifford)
  - Read quorum checks for identical version numbers, write quorum ask for allowance for an update
- Multiple Copy Update
  - cf. problems in databases (e.g., Bernstein, Goodman, 1984)

Agenda

Network File System (NFS)

Design Goals (1985)

- Sharing in a network of heterogeneous systems
  - Starting point: Diskless workstations
- Access transparency
  - No particular path names, libraries, or recompilation
- Portability
  - Definition of NFS as interface
  - Implementation of client and server side may be differ
- Simple handling of site failures
  - Statelessness of server
- Performance
  - Equivalent to local disk access
- Industry standard
  - By interface disclosure and reference implementation

Overall Architecture

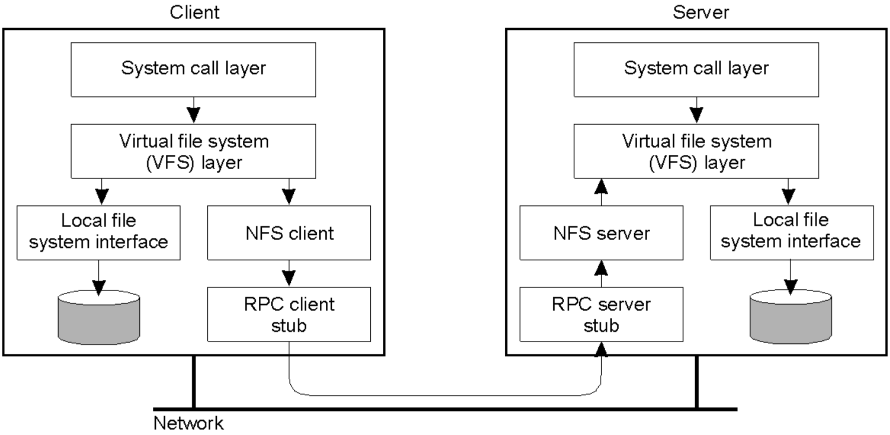


Fig. from Tanenbaum/Steen

Overall Architecture

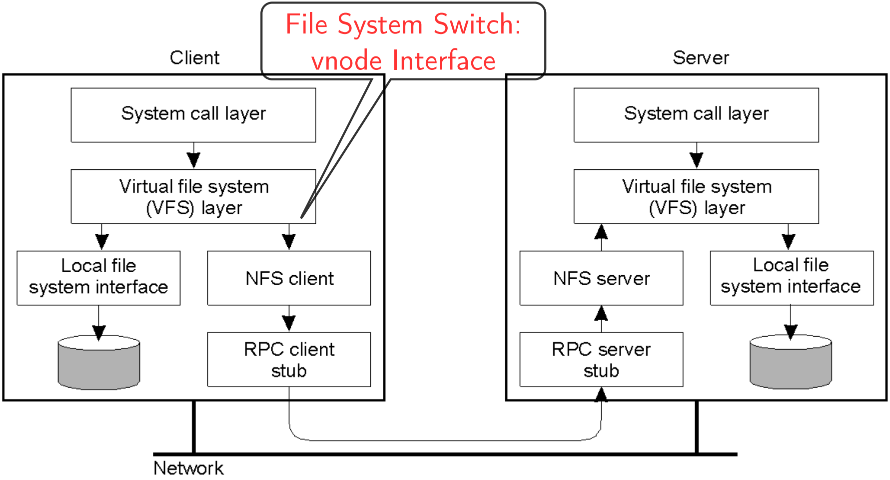


Fig. from Tanenbaum/Steen

Operating Principle

Roles

- Each node can be client and server simultaneously
- Each NFS server exports one or multiple directories (including the entire subtree)
- Common access by multiple clients is possible
- Client access requires mounting

Naming

- Hierarchical UNIX file namespace
- Location transparency is accomplished only by convention
  - Not enforced
  - Mountpoints can in principle be named arbitrarily

Locating

- Local mount table in the OS

⇒ No protocol for locating required

Directory Structure

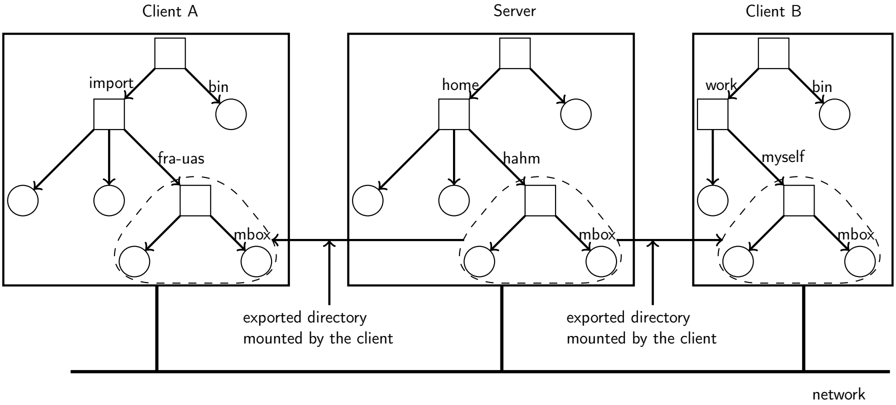


Fig. after Tanenbaum/Steen

Directory Structure

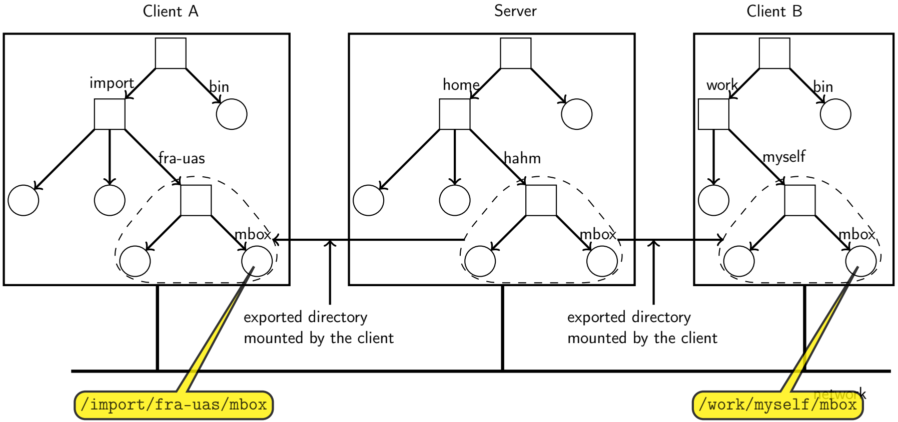
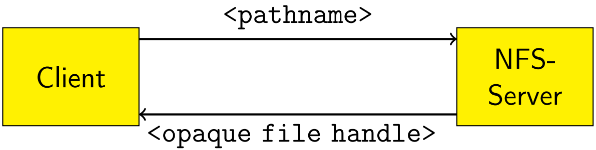


Fig. after Tanenbaum/Steen

Mount Protocol

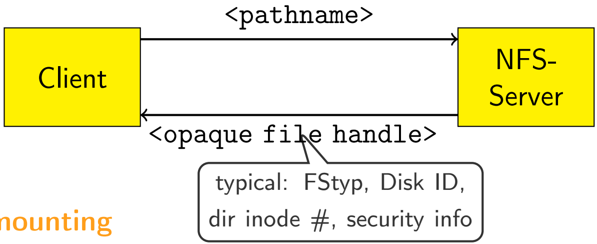
- Exists as subprotocol until version 3
- Integrated into the general access protocol since version 4



- **Static mounting**
  - Happens at boot time
- Problem:
  - Under certain circumstances server is not available at the time of mounting
  - → Client cannot boot without problems

Mount Protocol

- Exists as subprotocol until version 3
- Integrated into the general access protocol since version 4



- Static mounting
  - Happens at boot time
- Problem:
  - Under certain circumstances server is not available at the time of mounting

  → Client cannot boot without problems

Automounter

Introduced to solve problems of static mounting

Operating Principle

- Mapping:
  local mountpoint ↔ set of exported directories
- No action at boot time
- First access below the mountpoint causes a message to each servers in the set
- Who replies first, gets mounted
  - Failing server do not respond and can be tolerated
  - Load balancing is possible
- No support for general replication

  ⇒ Often only used for read-only file systems (e.g., `/usr`)

Access Protocol: Differences between Version 3 and 4

For access to directories and files, analog to *UNIX system calls*

Differences between version 3 and newer version 4

- Version 3 is stateless
  - No support for `open` and `close`
  - `read`/`write` have to provide required environment (`file handle, offset, nbytes`)
  - No file locks, only via separate lock server
- Version 4 is not stateless!
  - Goal: Allow for efficient use of NFS for WANs
  - Requires efficient client-side caching
  - Impossible to solve stateless
  - File locks are possible

Access Protocol: RPC

Underlying Protocol

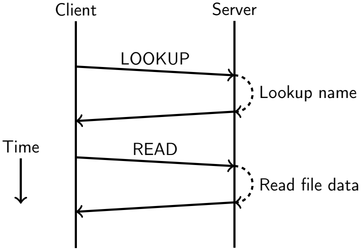
- SunRPC (ONC RPC) with XDR data encoding
- at-least-once-semantics
- Uses UDP/IP

Service Interface

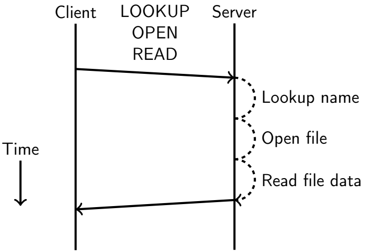
Operation	v3	v4	Description
create	Yes	No	Create a regular file
create	No	Yes	Create a nonregular file
link	Yes	Yes	Create a hard link to a file
symlink	Yes	No	Create a symbolic link to a file
mkdir	Yes	No	Create a subdirectory in a given directory
mknod	Yes	No	Create a special file
rename	Yes	Yes	Change 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

Access Protocol: Compound Operations

- Performance improvement in version 4
- Particularly relevant for WANs
- No concurrency control or atomicity



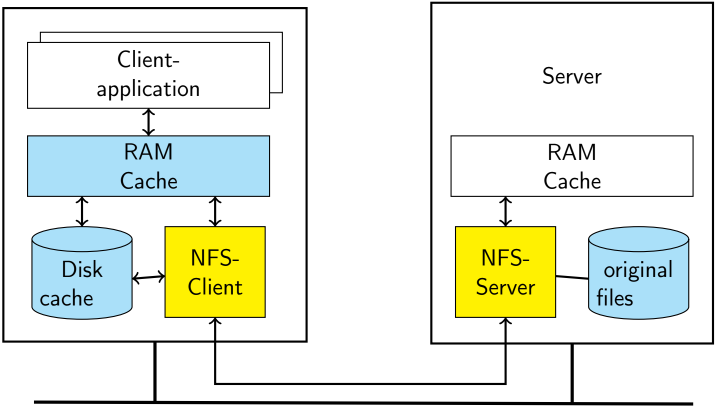
until Version 3



Version 4

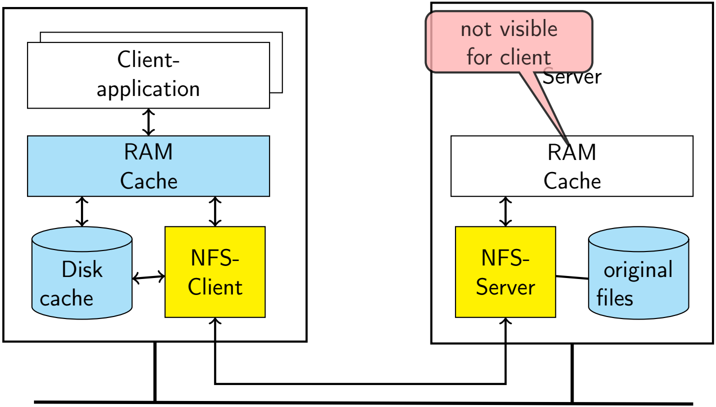
Caching

Client side caching



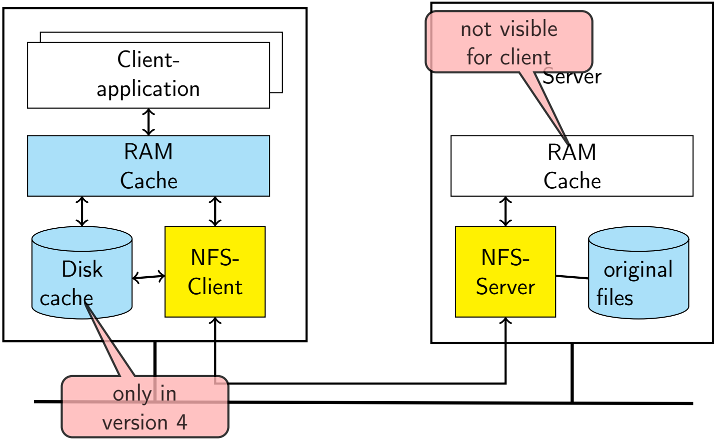
Caching

Client side caching



Caching

Client side caching



RAM Cache

- Caching individual blocks of remote files
- Big block size for efficient transfer, typical 8 KB
- Read-ahead for the next block
- Access to executable files with *size* < *thresshold* results in complete transfer

Cache Coherency

- Not given for version 3
  - Problem: Multiple clients may cache or even modify blocks of the same file/directory
  - **Timestamp based** weak validation scheme
    - Validation on `open()`, cache miss and timeout (typical: files 3 s, directories 30 s)
    - After checking its validity this assumed for a certain **duration**
    - Write-through for blocks of directories
    - All modified blocks are transferred to the server at latest on `close()`
  - Cache may contain outdated files and directories

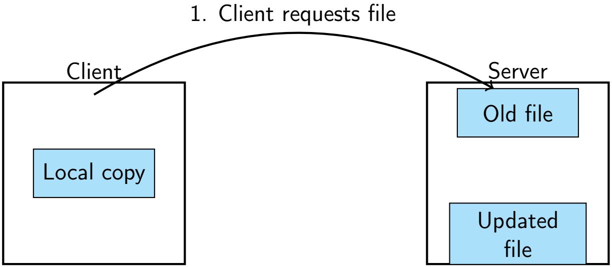
  ⇒ Cooperation of processes via file system not always correct for NFSv3
- Given for version 4
  - **Cache invalidation** of outdated files, checked on `open()`
  - Session semantics

Caching and File Delegation

- Only for version 4
- **Delegation** of server tasks to the client. This checks `open()` and `close()` operations of other clients
- Possibility to **revoke** delegation is required

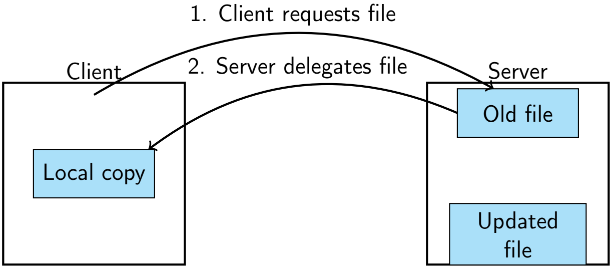
Caching and File Delegation

- Only for version 4
- **Delegation** of server tasks to the client. This checks `open()` and `close()` operations of other clients
- Possibility to **revoke** delegation is required



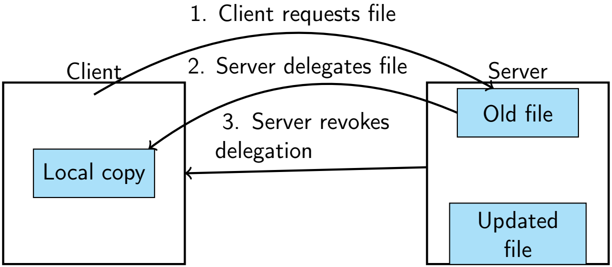
Caching and File Delegation

- Only for version 4
- **Delegation** of server tasks to the client. This checks `open()` and `close()` operations of other clients
- Possibility to **revoke** delegation is required



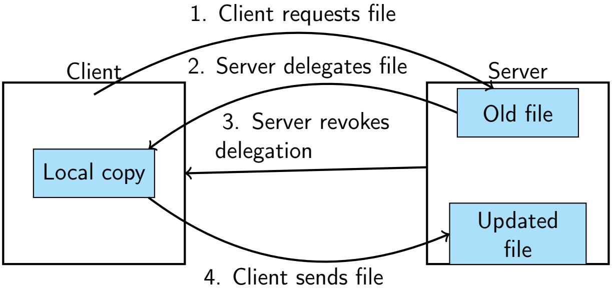
Caching and File Delegation

- Only for version 4
- **Delegation** of server tasks to the client. This checks `open()` and `close()` operations of other clients
- Possibility to **revoke** delegation is required



Caching and File Delegation

- Only for version 4
- **Delegation** of server tasks to the client. This checks `open()` and `close()` operations of other clients
- Possibility to **revoke** delegation is required



Security

Principal Architecture

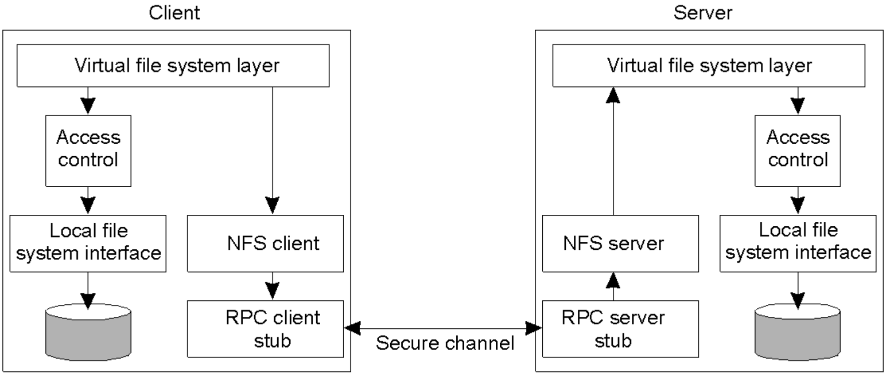


Fig. from Tanenbaum/Steen

Security

Principal Architecture

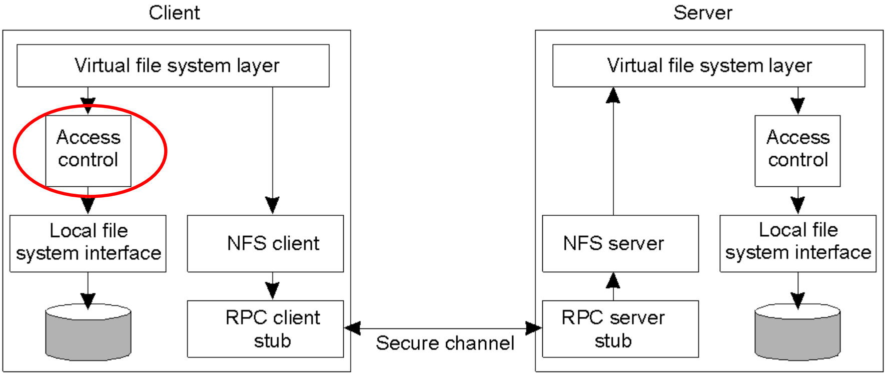


Fig. from Tanenbaum/Steen

Security

Principal Architecture

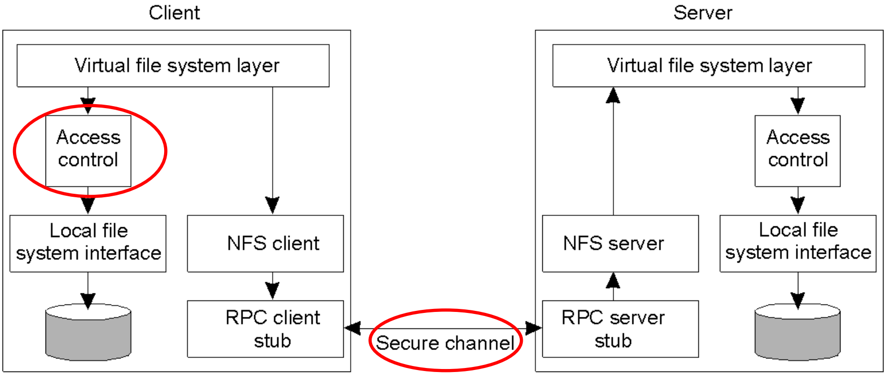


Fig. from Tanenbaum/Steen

Security

Principal Architecture

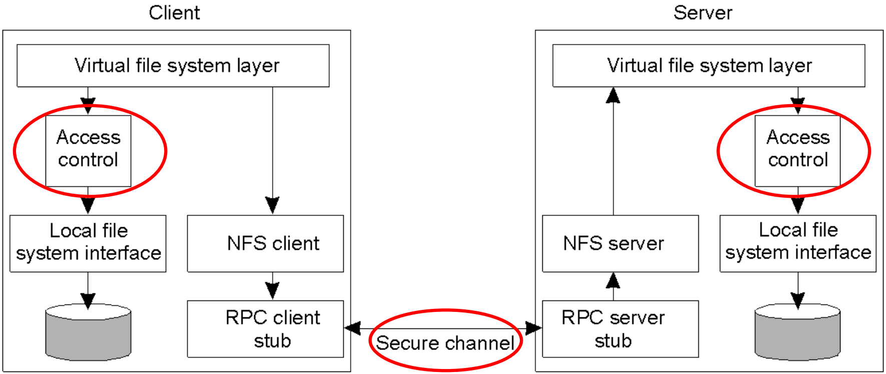


Fig. from Tanenbaum/Steen

Secure RPCs

in version 3:

- Only authentication
  - System
    - Based on UNIX IDs (`uid, gid`)
    - Transferred unencrypted without signature (server trusts client)
  - Diffie-Hellman
    - rarely used
    - Nowadays considered insecure because of too short key length
  - Kerberos

in version 4:

- No built-in mechanisms
- Supports `RPCSEC_GSS`
  - Security environment for various mechanisms that can be hooked in
  - Besides authentication it supports integrity and confidentiality

Secure RPCs for Version 4

Architectures of secure RPCs in version 4:

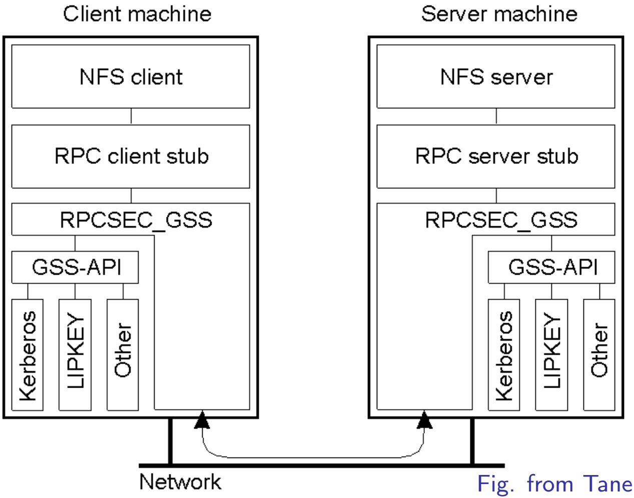


Fig. from Tanenbaum/Steen

Access Control

in version 3:

- **UNIX permission** checking (uid, gid) at server side

in version 4:

- **ACL based**
- Subjects strongly differentiated

Access Control Operations

Operation	Description
read_data	Permission to read the data contained in a file
write_data	Permission to modify a file's data
append_data	Permission to append data to a file
execute	Permission to execute a file
list_directory	Permission to list the contents of a directory
add_file	Permission to add a new file to a directory
add_subdirectory	Permission to create a subdirectory to a directory
delete	Permission to delete a file
delete_child	Permission to delete a file or directory within a directory
read_acl	Permission to read the ACL
write_acl	Permission to write the ACL
read_attributes	The ability to read the other basic attributes of a file
write_attributes	Permission to change the other basic attributes of a file
read_named_attrs	Permission to read the named attributes of a file
write_named_attrs	Permission to write the named attributes of a file
write_owner	Permission to change the owner
synchronize	Permission to access a file locally at the server with synchronous reads and writes

Access Control Subjects

User type	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

According to Tanenbaum/Steen

Agenda

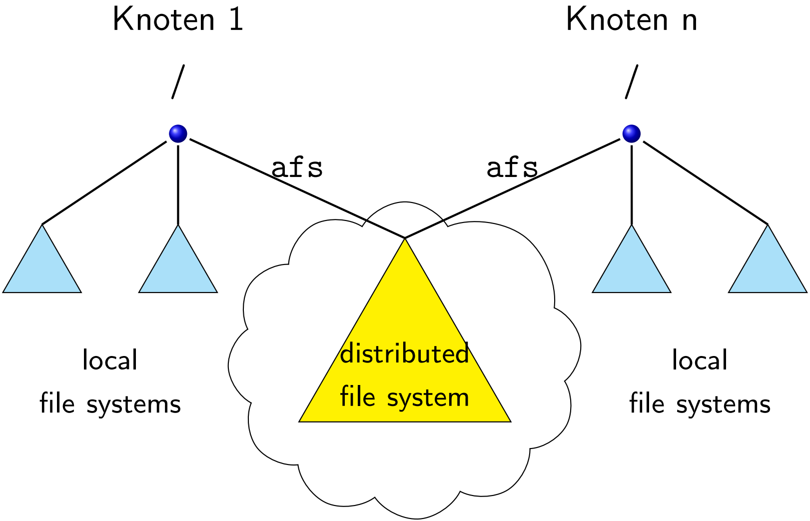
Andrew File System (AFS)

- Carnegie Mellon University (CMU) with IBM, 1983 – 1989, later Transarc and OpenAFS
- File system for the campus with more than 5,000 active students
- Goals
  - **Location transparency**, common **global file namespace**, accessible via the local name `/afs`
  - High **performance**
  - High **availability**
    - Replication
  - High **security**
    - Secure authentication
    - Encrypted transfer
    - ACLs for access control
  - Automatic migration of home directories of users

Coda

Newer versions of AFS-2

File Namespace



Overall Architecture

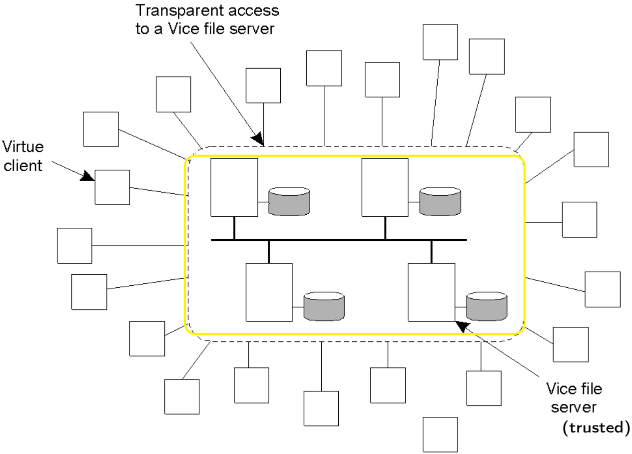
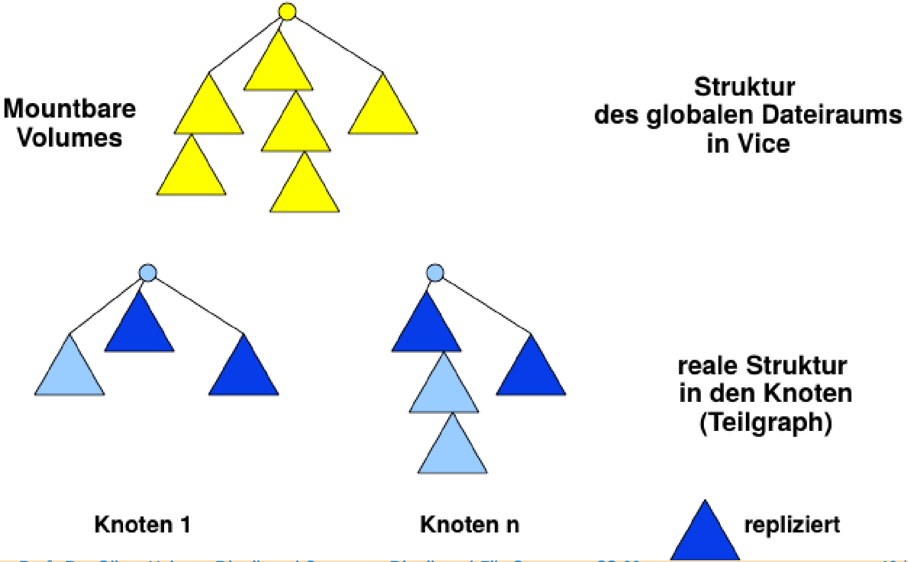


Fig. from Tanenbaum/Steen

Volumes


Mountbare
Volumes
Struktur
des globalen Dateiraums
in Vice
reale Struktur
in den Knoten
(Teilgraph)
Knoten 1
Knoten n
repliziert

Virtue Client Architecture

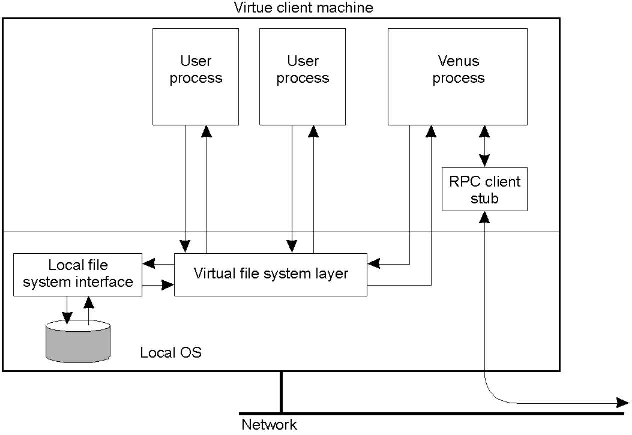
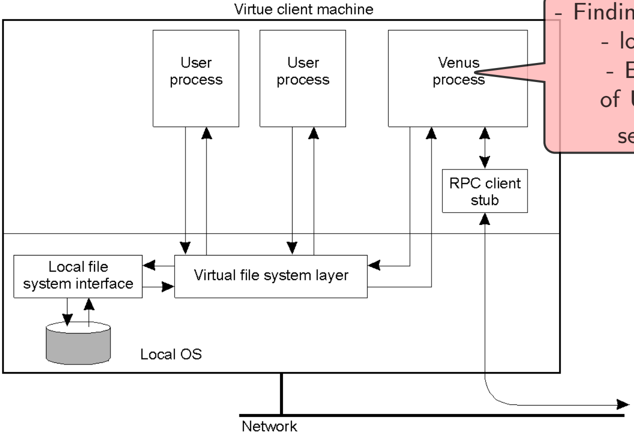


Fig. from Tanenbaum/Steen

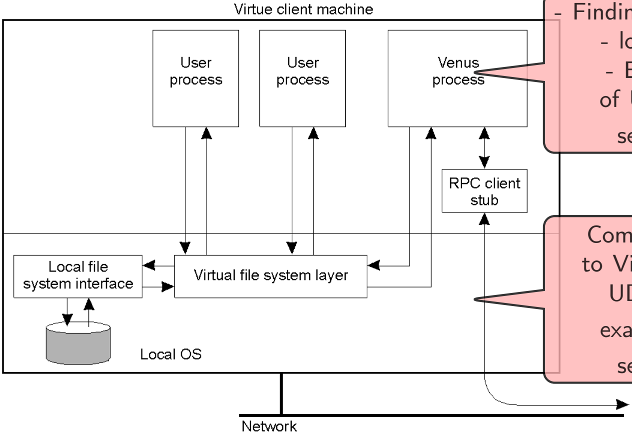
Virtue Client Architecture



- Finding files in Vice
  - local cache
  - Emulation of UNIX file-semantics

Fig. from Tanenbaum/Steen

Virtue Client Architecture



- Finding files in Vice
  - local cache
  - Emulation of UNIX file-semantics

Communication to Vice via RPC UDP based exactly-once-semantics

Fig. from Tanenbaum/Steen

Properties

- Commonly used files with **session semantics**
- Local caching: of entire files until AFS-2/of big file blocks (64 kB) since AFS-3
- **Cache coherency**
  - Check the cache validity not required for each `open()`
  - Callback procedure, i.e., explicit **invalidation** by the server before another client gets write permissions

Security

- **Organisation**
  - Vice-Server are trusted
  - No client applications on servers
  - Introduction of administrative cells to increase scalability
- **Subjects**
  - User
  - Groups
- **Authentication**
  - Particular authentication server, Kerberos (since AFS-3)
  - Secure RPC
- **Access Control**
  - ACLs defined for directories, apply for all files of the directory

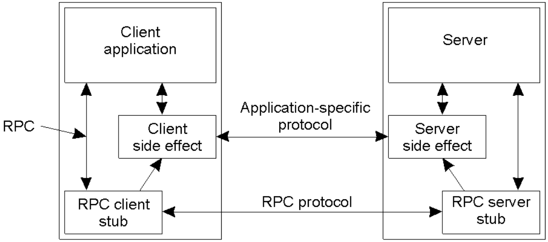
Coda

- Further development of AFS-2 since 1987
- **Goals:**
  - High availability of the files
  - Client can continue to work, even when the server is temporarily not reachable (**network partioning**)
  - Inclusion of **mobile computers** (intentional network partioning)

Communication

RPC2-System

- Use of internal multi-threading for Venus and Vice
- Support for so called side effects



- MultiRPCs (transparent call to multiple servers) used for cache invalidation
  (parallel unicast calls or use of IP multicast)

File IDs

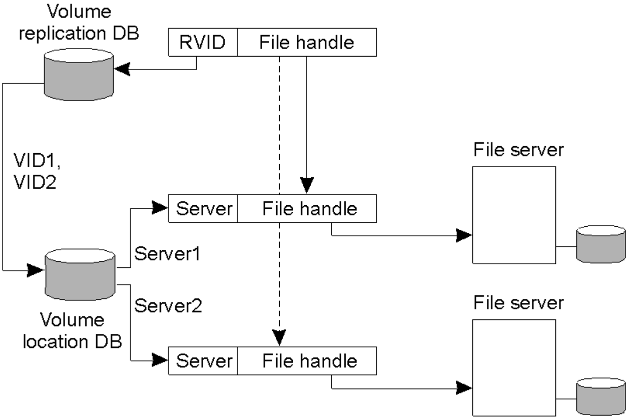


Fig. from Tanenbaum/Steen

File IDs

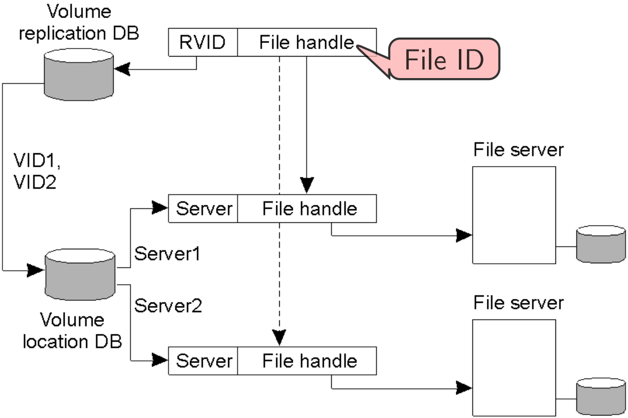


Fig. from Tanenbaum/Steen

File IDs

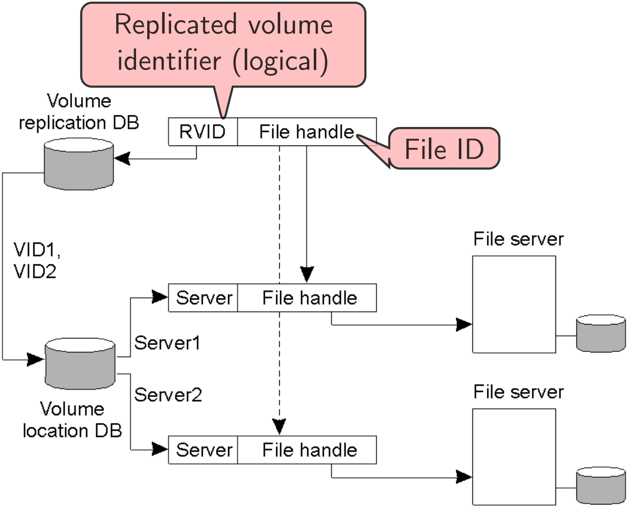


Fig. from Tanenbaum/Steen

File IDs

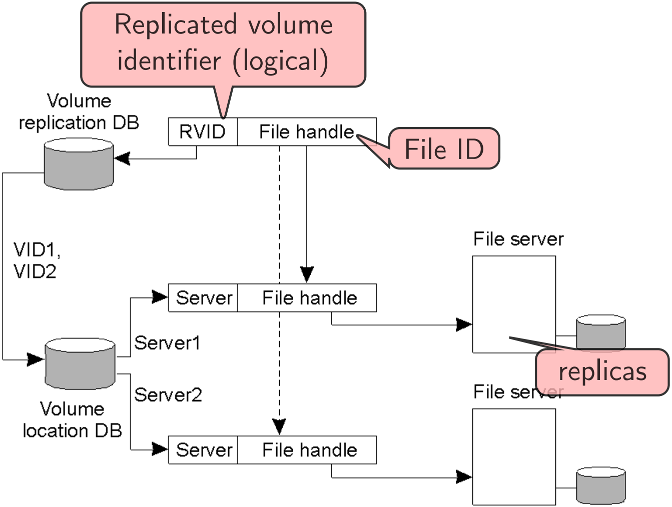


Fig. from Tanenbaum/Steen

Properties for Normal Operation

- Session semantics
- One writer or multiple readers
- Transaction behaviour

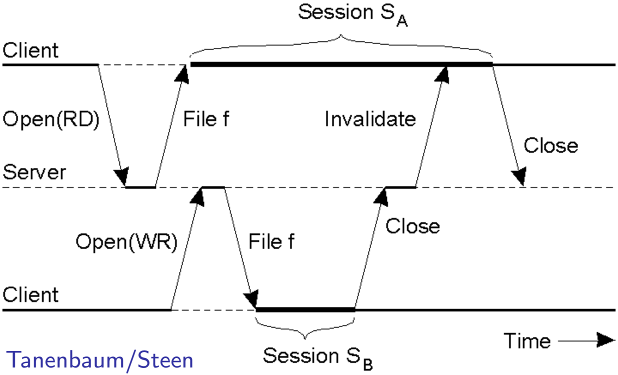


Fig. from Tanenbaum/Steen

Properties for Normal Operation

- Session semantics
- One writer or multiple readers
- Transaction behaviour

noncritical: A reads from older file version, processes this

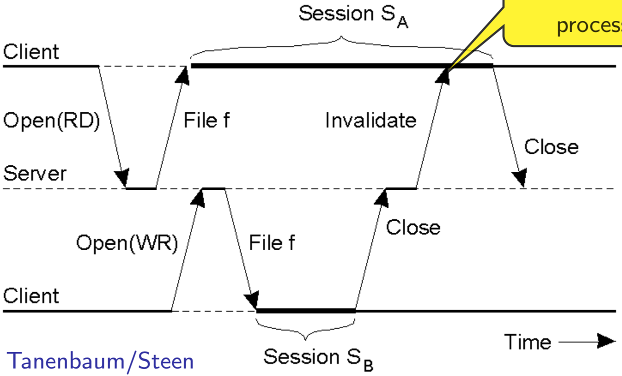


Fig. from Tanenbaum/Steen

Caching

- On open() the file is loaded into the **client's cache**
- Server makes **callback** promise
- For **invalidation** the server sends a callback break

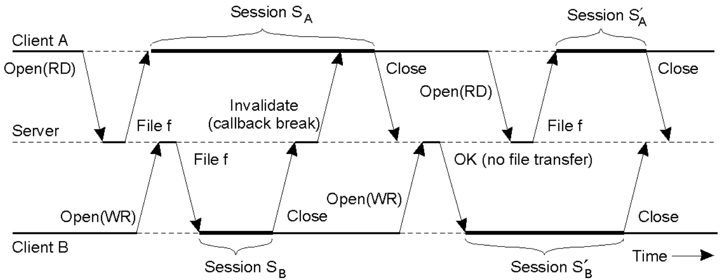


Fig. from Tanenbaum/Steen

Caching

- On `open()` the file is loaded into the **client's cache**
- Server makes **callback** promise
- For **invalidation** the server sends a callback break

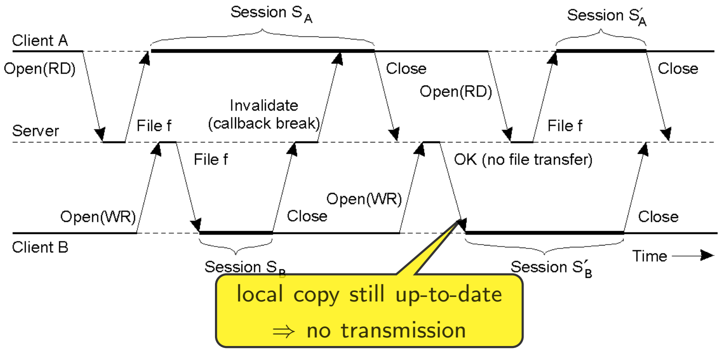


Fig. from Tanenbaum/Steen

Caching

- On `open()` the file is loaded into the client's cache
- Server makes callback promise
- For invalidation the server sends a callback break

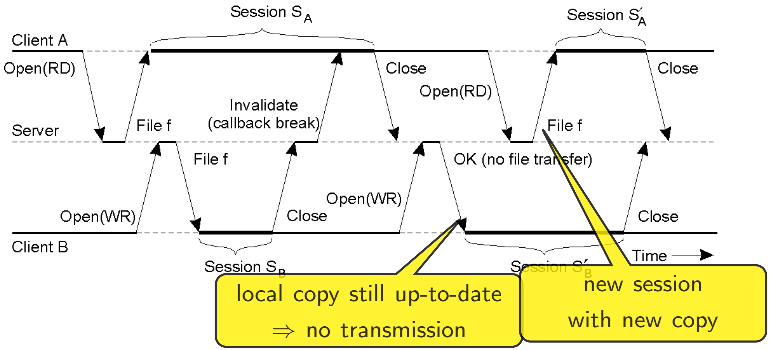


Fig. from Tanenbaum/Steen

Server Replication and Network Partitioning

- Volume is the unit for replication
- Volume Storage Group (VSG)
  - Set of servers with a copy of a volume
- Accessible Volume Storage Group (AVSG)
  - Subset of VSG which can be accessed from the client
- Reading from a replica, write to all via MultiRPC
- Optimistic strategy for file replication
  - On partitioning multiple writers may exist and write back to their respective AVSGs
- Maintaining of version vectors ($\rightarrow$ vector timestampes), check on update
- Merge of multiple versions later on, may require manual assistance

Connectionless Operation

- Connectionless: AVSG$= \emptyset \Rightarrow$ use of local copy
- Conflict detection on transmission to server
- Observation
  - Conflicts are rare since modifications to one file by multiple processes are infrequent
- Problem
  - Keep relevant files in local cache when disconnect happens
- Approach: Hoarding (of files)
  - Heuristic method
  - Explicit declaration of files and directories by the user
  - Prioritisation by matching with current access information
  - Cache alignment (hoard walk) every 10 minutes
  - Good experiences, but of course it happens that occasionally relevant files are missing

Summary

Issue	NFS	Coda
Design goals	Access transparency	High availability
Access model	Remote	Up/Download
Communication	RPC	RPC
Client process	Thin/Fat	Fat
Server groups	No	Yes
Mount granularity	Directory	File system
Name space	Per client	Global
File ID scope	File server	Global
Sharing sem.	Session	Transactional
Cache consist.	write-back	write-back
Replication	Minimal	ROWA
Fault tolerance	Reliable comm.	Replication and caching
Recovery	Client-based	Reintegration
Secure channels	Existing mechanisms	Needham-Schroeder
Access control	Many operations	Directory operations

According to Tanenbaum/Steen

Agenda

Storage Networks

Use of network technology and distributed systems in a storage system

Motivation

- Cost reduction
  - Reduced provided storage capacities
  - Central administration
- More flexible provisioning
  - Faster adaption to changing requirements
- Scalability
  - From small to very large storage capacities
- Options for disaster recovery
  - Data mirroring at remote locations

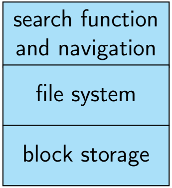
Architecture Approaches

Today's essential architectural approaches

- Direct Attached Storage (DAS) (traditional local storage)
- Storage Area Networks (SAN)
- Network-Attached Storage (NAS)
- Content Addressed Storage (CAS)

Storage Systems as Layered Systems

Basic breakdown of the data storage functionality

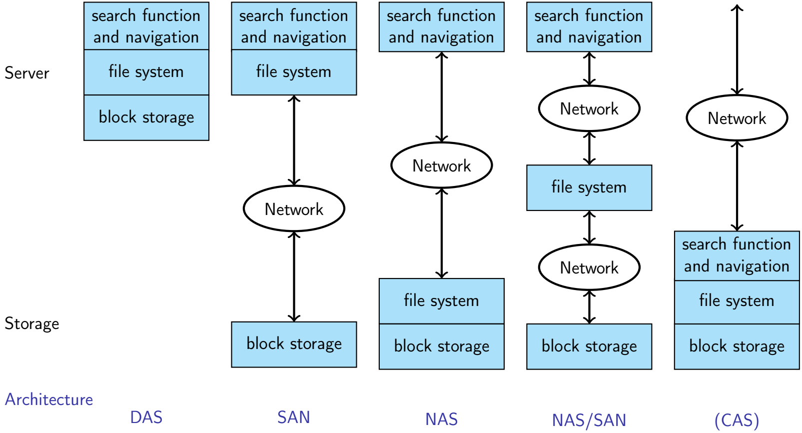


path names, query, index, metadata, ...

Mapping to logical set of blocks,
e.g., NFS, AFS, Microsoft SMB/CIFS

Mapping to phys. storage device

Overview for Integration of Networks



Direct Attached Storage (DAS)

- Traditional, locally attached storage devices
- Local OS contains file system and device driver
- Typical device interfaces
  - IDE/ATA, SCSI, Serial ATA (SATA), ...
- Limitations
  - Number of available channels
  - Number of attachable devices per channel
  - maximum distance ($\approx 1 - 25$ m)

  $\Rightarrow$ No disaster recovery possible
  - Performance

  $\Rightarrow$ Not very scalable

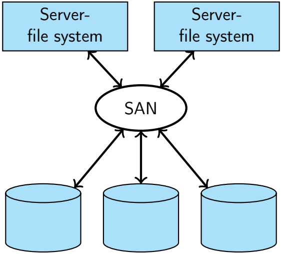
Storage Area Network (SAN)

Operating principle

- SAN provides **block storage** (e.g., hard disks as logical devices)
- Server operating systems provide one or multiple file system
- Block storage is accessed by the servers via SAN

Advantages

- Very simple **extensibility**
- Very high degree of flexibility in **assignability**
- Very **scalable**
- Basis for replication, also for disaster recovery
- Bootable network partitions
- Especially suited for applications who works with volumes (without file system), e.g., DBMS



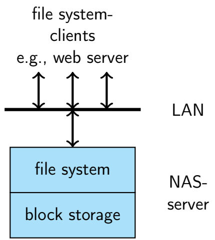
Network Attached Storage (NAS)

Operating principle

- NAS provides **network file systems** (e.g., NFS, SMB/CIFS)
- Clients may use these file systems

Advantages

- Storage **consolidation**
- **Expandability**
- **Scalability**
- **Manageability**
- Especially suited for applications which are based on file access, e.g., web applications or home directories



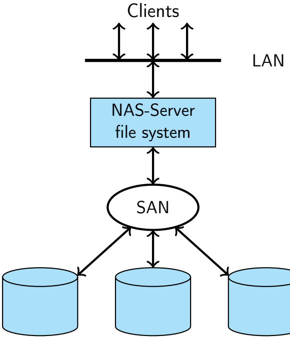
NAS Examples

- Home server
  - Synology
  - Zyxel
- IBM Storvize V7000

Combination of NAS and SAN

Operating principle

- NAS and SAN can be used combined
- Advantages can be combined



Content Addressable Storage (CAS)

Main idea: Immutable information

- ⇒ also called **Fixed Content Storage (FCS)**
- Goal: archive storage

Subproblem of the Information Lifecycle Management (ILM)

- Mass data (hundreds of terabytes or even petabytes)
- Longevity
- Integrity
- Immutability of documents, partly required by law

Usage for content management

- Digital media (audio or image documents)
- Email archiving
- Health care (X-ray images etc.)
- ...

CAS: Operating Principle

- **Content identifier** as reference
- Determination of a content identifier
  - Only by its content ($\rightarrow$ hash value) $\Rightarrow$ location independent
  - Or by its storage location (inverted)
- System determines location for access via its content identifier

Examples

- First System: EMC Centera 2002
- iTernity iCAS (Software solution)
- Various open source solutions, e.g., Keep Content Addressable Storage

Important takeaway messages of this chapter

- Distributed file systems enable concurrent access access to files by different users, independent of their storage location.
- Access to files can lead to different results depending on the consistency semantics. consistency semantics.
- By dividing the functionality for information storage on different systems in the network, scalability, fault tolerance, etc. can be improved.