

Verteilte Systeme

R. Kaiser, R. Kröger, O. Hahm

(HTTP: <http://www.cs.hs-rm.de/~kaiser>

E-Mail: robert.kaiser@hs-rm.de)

Kai Beckmann

Sebastian Flothow

Sommersemester 2022

9. Verteilte Dateisysteme



<https://tagebucheindeutschen.wordpress.com/2016/08/06/neues-von-der-scheinbehoerden-front-strategiewechsel-der-firma-finanztmt/comment-page-1/>

Inhalt



9. Verteilte Dateisysteme

9.1 Einführung

9.2 Grundlagen

9.3 NFS

9.4 AFS und Coda

9.5 Speichernetze

Einführung



Datenhaltungssysteme

| | Dateisysteme | Datenbank-systeme | Objekt-management-systeme |
|---|---------------------------------------|--|--|
| Inhalt | universell | Massendaten weniger struktureller Typen | Beziehungen stehen im Vordergrund |
| Gespeicherte Informationen | passiv | passiv | aktiv |
| Semantik auf- prägender Code | extern | extern | intern (Typen) |
| Zugriff | über Namen, einfache Navigation | komplexe assoziative Suchfunktionen | komplexe Such- und Navigations- funktionen |

Modelle von Dateisystemen



Dateien als klassische Abstraktion in Betriebssystemen

Historische Entwicklung im folgenden betrachtet

1 Rechner, 1 Benutzer, 1 Prozess

- zu lösende Probleme
 - ▶ Struktur des Dateisystems
 - ▶ Benennung (Naming)
 - ▶ Programmierschnittstelle
 - ▶ Abbildung auf physikalischen Speicher
 - ▶ Integrität
- Beispiele
 - ▶ PC-DOS
 - ▶ klassisches MacOS

Modelle von Dateisystemen (2)



1 Rechner, 1 Benutzer, mehrere Prozesse

- zusätzlich zu lösende Probleme
 - ▶ Nebenläufigkeitskontrolle
- Beispiel
 - ▶ OS/2

1 Rechner, mehrere Benutzer, mehrere Prozesse

- zusätzlich zu lösende Probleme
 - ▶ Sicherheit und Rechteverwaltung
- Beispiel
 - ▶ UNIX

mehrere Rechner, mehrere Benutzer, mehrere Prozesse

- im weiteren betrachtet
- zusätzlich zu lösende Probleme
 - ▶ Verteiltheit, d.h.
 - ★ sichtbare Gesamtstruktur
 - ★ Zugriffsmodell
 - ★ Aufenthaltsort
 - ★ Replikation
 - ★ Verfügbarkeit
 - ★ ...
 - ▶ Kein Zugriff auf gemeinsamen Block-Speicher der Knoten (shared nothing)
 - ▶ Sharing gemeinsamer Platten zwischen den Knoten hier nicht weiter betrachtet
(vgl. 9.5: SAN Storage Area Networks) → Cluster File Systems
- Client/Server-Modell
 - ▶ ausgezeichneter File-Server
- Peer-to-Peer-Modell
 - ▶ jeder kann Dateien bereitstellen

Historische Vorläufer

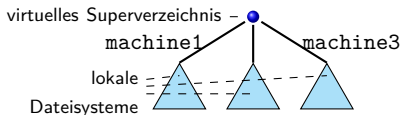


Völlige Trennung

- Ausschließlich lokale Zugriffe
- Dateitransfer zwischen isolierten Dateisystemen (Download/Upload-Modell)
- Beispiel: UNIX uucp, ftp, rcp, scp

Adjungierte Dateisysteme

- Zugriff auf entfernte Dateien
- Explizite Angabe des Aufenthaltsorts im Dateinamen
- Beispiel: Newcastle Connection



```
/machine1/<localpath>  
machine2!<localpath>  
/../machine3/<localpath>
```

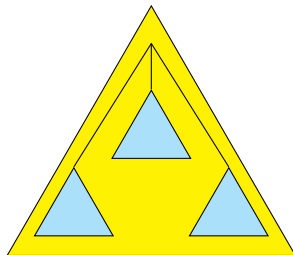

Verteiltes Dateisystem



Ein *verteiltes* Dateisystem ist ein Dateisystem, das seinen Nutzern auf allen Maschinen im Netz ein einheitliches Dateisystem zur Verfügung stellt

Mögliche Transparenzarten:

- Ortstransparenz
 - ▶ Dateiname enthält keine Ortsangabe
- Zugriffstransparenz
 - ▶ API unabhängig, ob Datei lokal oder entfernt
- Namenstransparenz
 - ▶ Dateiname an allen Stellen identisch
- Replikationstransparenz
- ...



Typische Designziele



- Hohes MaSS an Transparenz
 - ▶ s.o.
- Performance
 - ▶ ähnlich lokalem Zugriff
- Hohe Verfügbarkeit (Availability) / Fehlertoleranz
- Sicherheit
- Skalierbarkeit
- Unterstützung für mobile Knoten mit zeitweiser Diskonnektivität
- Unterstützung für shared disk und shared nothing (lose gekoppelte) Knoten
- Cloud-Anbindung

Lösungsüberblick



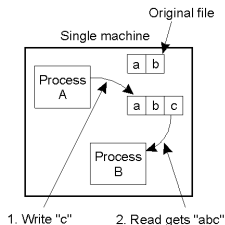
- Network File System (NFS) — ab 1985
- Andrew File System (AFS) + Coda — ab etwa 1985
- Common Internet File System (CIFS) + Server Message Block (SMB)
- GlusterFS (Gluster Inc. → Red Hat 2011)
- IBM General Parallel File System (GPFS) (ursprünglich Cluster File System, weiterentwickelt)
- Google File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

Grundlagen

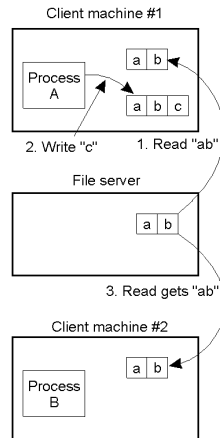


Zugriffskonsistenzproblem

- (a): Änderungen für jeden unmittelbar sichtbar und damit aktuell
- (b): sichtbare Werte können veraltet sein



(a)



(b)

Abb. aus Tanenbaum/Steen

Semantiken



Strenge Konsistenz

- Änderungen unmittelbar für alle anderen sichtbar
- Beispiel: lokales UNIX

Sitzungssemantik

- Aktualisierung der Datei beim Schließen
- erlaubt lokalen Cache, solange Datei geöffnet ist
- Beispiel: Andrew File System

Read-Only Dateien

- Änderungen sind nicht möglich
- Gemeinsame Nutzung und Replikation werden deutlich vereinfacht

Transaktionssemantik

- Änderungen auf einer Menge von Dateien finden atomar statt (vgl. Kap. 8)

Zustandslose / zustandsbehaftete Server



Zustandlosigkeit

- Server hat kein Gedächtnis

Vorteile zustandsloser Server

- Recovery leicht realisierbar
- Keine Probleme mit Client-Abstürzen
- Öffnen / SchlieSSen von Dateien unnötig
- Anzahl offener Dateien unbegrenzt

Vorteile zustandsbehafteter Server

- kürzere Nachrichten
- höhere Performance
- Read-ahead möglich
- Idempotenz von Operationen leichter realisierbar
- Dateisperren möglich

Replikation



Ziele

- Verfügbarkeit der Daten
- Lastausgleich
- Transparenz aus Benutzersicht

Übliche Algorithmen

- hier nicht im Detail behandelt
- Primary Copy Update
 - ▶ Primary = ausgezeichnete Kopie
 - ▶ Veränderungen nur auf Primary, dieser kümmert sich um Kopien
 - ▶ Primary sieht damit alle Updates
 - ▶ Lesen von beliebiger Kopie (Performance)
- Voting (Gifford)
 - ▶ Lese-Quorum und Schreib-Quorum haben einen nicht-leeren Schnitt
- Multiple Copy Update
 - ▶ vgl. Problematik in Datenbanken (z.B. Bernstein, Goodman, 1984)

Network File System (NFS)



Designziele (1985)

- Sharing in einem Netzwerk heterogener Systeme
 - ▶ Ausgangspunkt: Diskless Workstations
- Zugriffstransparenz
 - ▶ keine speziellen Pfadnamen, Bibliotheken oder Rekompilation
- Portabilität
 - ▶ Festlegung von NFS als Schnittstelle
 - ▶ Implementierung von Client- und Server-Seite kann unterschiedlich sein
- Einfache Behandlung von Stellenausfällen
 - ▶ Zustandslosigkeit des Servers
- Performance
 - ▶ äquivalent zu lokalem Plattenzugriff
- Industriestandard
 - ▶ durch Offenlegung von Schnittstelle und Referenz-Implementierung

Gesamtarchitektur

File System Switch:
vnode Interface

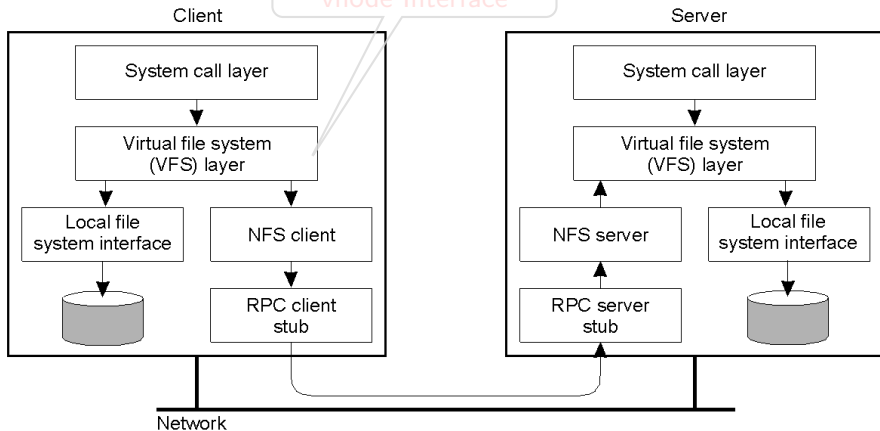


Abb. aus Tanenbaum/Steen

Gesamtarchitektur

File System Switch: vnode Interface

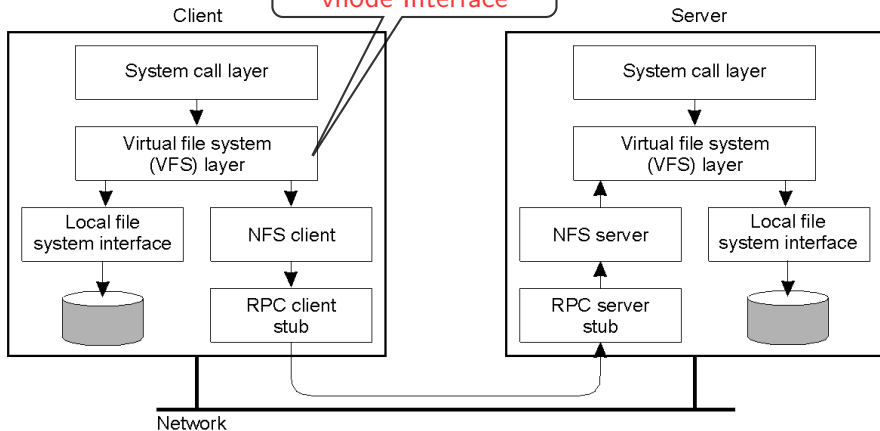


Abb. aus Tanenbaum/Steen

Funktionsweise

Rollen

- Jeder Knoten kann gleichzeitig Client und Server sein
- Jeder NFS-Server exportiert ein oder mehrere Verzeichnisse (mit dem gesamten Unterbaum)
- Gemeinsame Nutzung durch mehrere Clients möglich
- Client-Zugriff erfordert Mounting

Naming

- hierarchischer UNIX-Pfadnamensraum
- Ortstransparenz entsteht nur durch Konvention
 - ▶ nicht erzwungen
 - ▶ Mount-Punkte können prinzipiell beliebig benannt werden

Locating

- lokale Mount-Tabelle im Betriebssystem
- daher kein Protokoll zur Aufenthaltsortsbestimmung notwendig

Verzeichnisstruktur

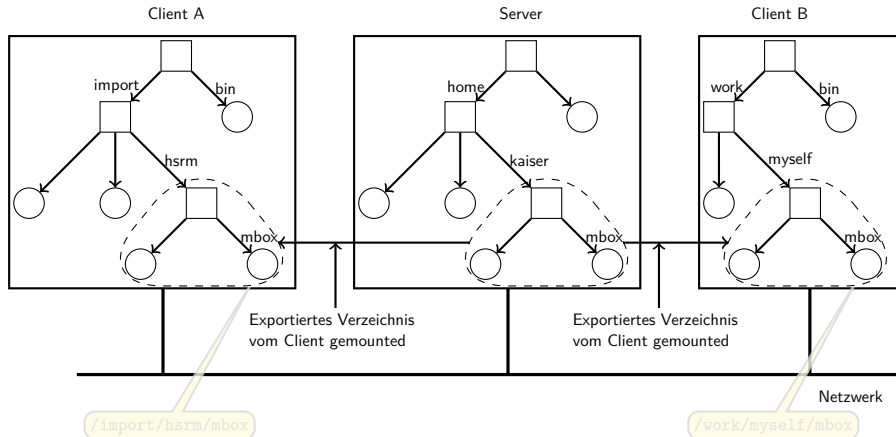


Abb. nach Tanenbaum/Steen

Verzeichnisstruktur

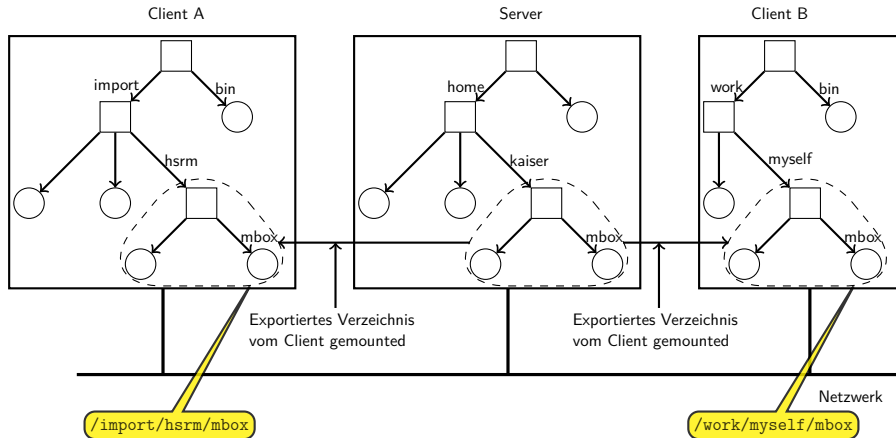


Abb. nach Tanenbaum/Steen

Mount-Protokoll

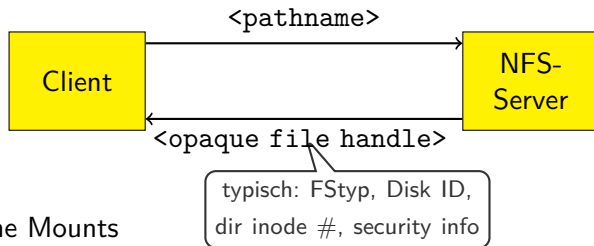
- existiert bis zur aktuellen Version 3 als Teilprotokoll
- in Version 4 in allgemeines Zugriffsprotokoll integriert



- Statische Mounts
 - ▶ zum Boot-Zeitpunkt ausgeführt
- Problem:
 - ▶ u.U. nicht-verfügbarer Server zum Mount-Zeitpunkt
 - Client kann nicht ohne Probleme booten

Mount-Protokoll

- existiert bis zur aktuellen Version 3 als Teilprotokoll
- in Version 4 in allgemeines Zugriffsprotokoll integriert



- Statische Mounts
 - ▶ zum Boot-Zeitpunkt ausgeführt
- Problem:
 - ▶ u.U. nicht-verfügbarer Server zum Mount-Zeitpunkt
 - Client kann nicht ohne Probleme booten

Automounter



Zur Lösung der Probleme statischer Mounts eingeführt Funktionsweise

- Zuordnung:
lokaler Mount-Punkt \leftrightarrow Menge von exportierten Verzeichnissen
- Keine Aktion zum Bootzeitpunkt
- Erster Zugriff unterhalb des Mount-Punkts bewirkt Nachricht an alle Server der Menge
- Wer zuerst antwortet, wird gemountet
 - ▶ Ausgefallene Server antworten nicht und können toleriert werden
 - ▶ Lastausgleich möglich
- Keine Unterstützung für allgemeine Replikation
 - ▶ daher oft nur für read-only-Dateisysteme genutzt (z.B. /usr)

Zugriffsprotokoll



Für Zugriffe auf Verzeichnisse und Dateien analog UNIX system calls

Unterschiede zwischen Version 3 und neuer Version 4

- Version 3 ist zustandslos
 - ▶ keine Unterstützung für open und close
 - ▶ read/write müssen notwendige Umgebung mit anliefern (file handle, offset, nbytes)
 - ▶ keine Sperren auf Dateien, sondern in separatem Lock-Server
- Version 4 ist **nicht** zustandslos !
 - ▶ Ziel: Effiziente Nutzung von NFS in Weitverkehrsnetzen ermöglichen
 - ▶ erfordert effizientes korrektes Caching auf Client-Seite
 - ▶ geht nicht zustandslos
 - ▶ auch Dateisperren werden erlaubt

Zugriffsprotokoll (2)



Unterlagertes Protokoll

- SunRPC (ONC RPC) mit XDR-Datenkodierung
- at-least-once-Semantik
- nutzt selbst UDP/IP

Dienstschnittstelle



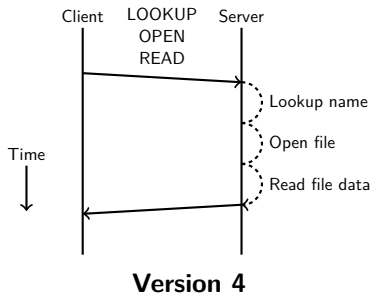
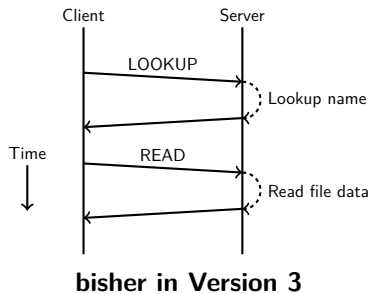
| Operation | v3 | v4 | Beschreibung |
|-----------|-----|-----|---|
| Create | Yes | No | Create a regular file |
| Create | No | Yes | Create a nonregular file |
| Link | Yes | Yes | Create a hard link to a file |
| Symlink | Yes | No | Create a symbolic link to a file |
| Mkdir | Yes | No | Create 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 |

Zugriffsprotokoll (3)



Zusammengesetzte Prozeduren

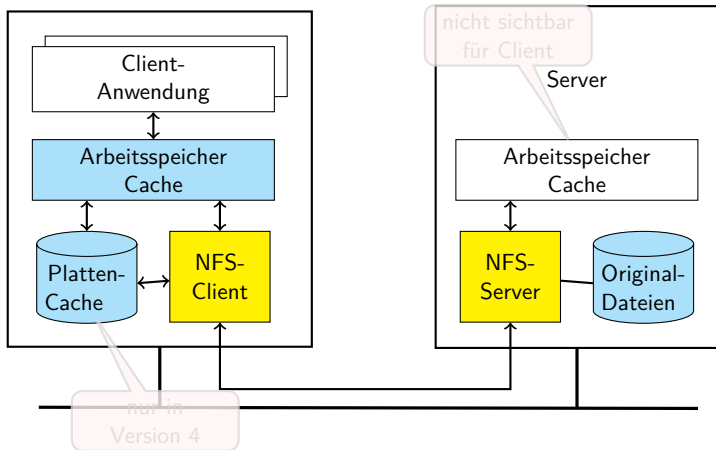
- Performance-Optimierung in Version 4
- insbesondere in Weitverkehrsnetzen relevant
- keine Nebenläufigkeitskontrolle oder Atomarität



Caching



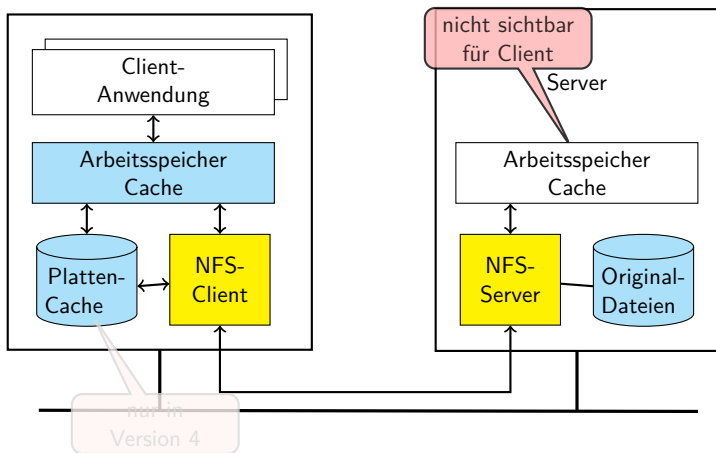
Client-seitiges Caching



Caching



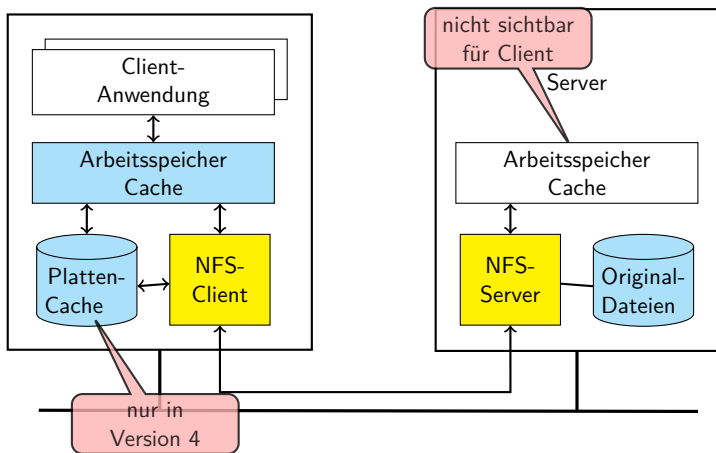
Client-seitiges Caching



Caching



Client-seitiges Caching



Caching (2)



Arbeitsspeicher-Cache

- Caching einzelner Blöcke entfernter Dateien
- große Blockgrößen für effizienten Transfer, typisch 8 KB
- Read-ahead des nächsten Blocks
- Zugriffe auf ausführbare Dateien mit $\text{Größe} < \text{Schwellwert}$ führt zu vollständiger Übertragung

Caching (3)



Cache-Kohärenz

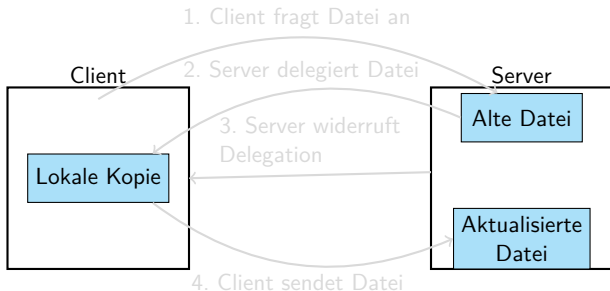
- in Version 3 **nicht** gegeben
 - ▶ Problem: Mehrere Clients können Blöcke derselben Datei / desselben Verzeichnisses cachen und auch modifizieren
 - ▶ zeitmarkenbasiertes unsicheres Validierungsschema
 - ★ Validierung bei `open()`, Cache Miss und Timeout (typisch: Dateien 3 Sek, Verzeichnisse 30 Sek)
 - ★ Nach Validierung wird Gültigkeit für eine Zeitspanne angenommen
 - ★ Write-Through für Blöcke von Verzeichnissen
 - ★ Alle modifizierten Blöcke werden spätestens bei `close()` an Server übertragen
 - ▶ Cache kann damit insgesamt veraltete Dateien und Verzeichnisse enthalten
- ⇒ Kooperation von Prozessen über Dateisystem unter NFS 3 nicht immer korrekt
- in Version 4 gegeben
 - ▶ Cache-Invalidierung veralteter Daten, Überprüfung bei `open()`
 - ▶ Sitzungssemantik

Caching (4)



File Delegation

- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft `open()`- und `close()`-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig

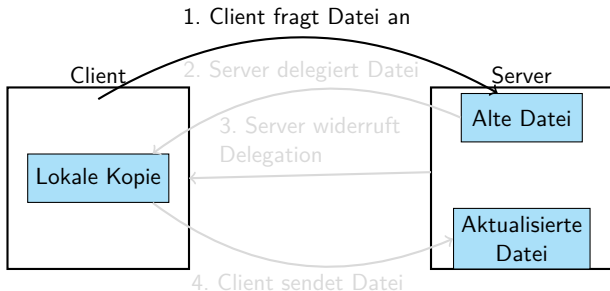


Caching (4)



File Delegation

- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft `open()`- und `close()`-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig

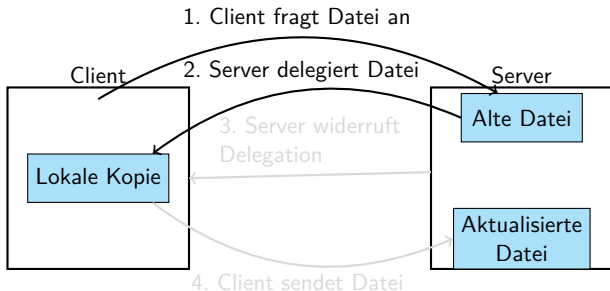


Caching (4)



File Delegation

- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft `open()`- und `close()`-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig

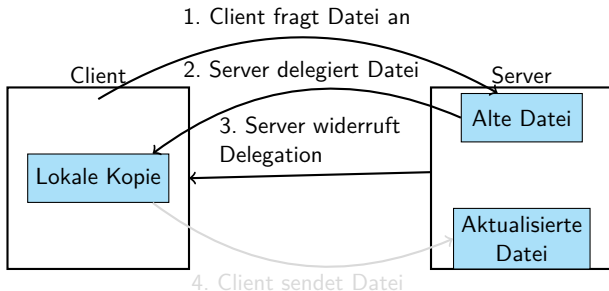


Caching (4)



File Delegation

- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft `open()`- und `close()`-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig

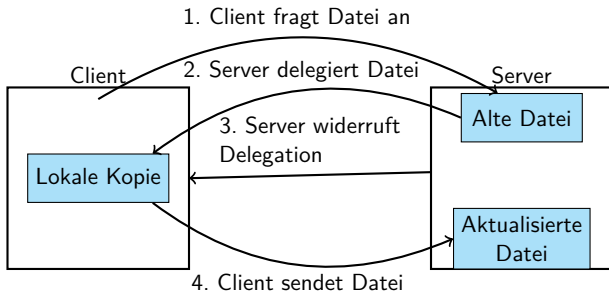


Caching (4)



File Delegation

- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft `open()`- und `close()`-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig



Sicherheit

Prinzipielle Architektur

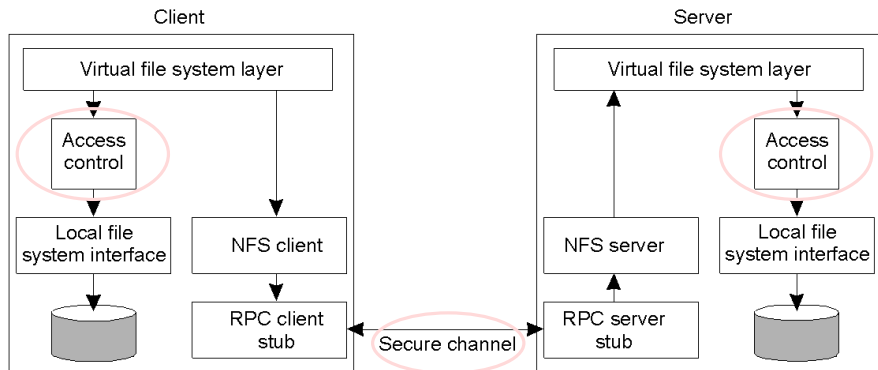


Abb. aus Tanenbaum/Steen

Sicherheit

Prinzipielle Architektur

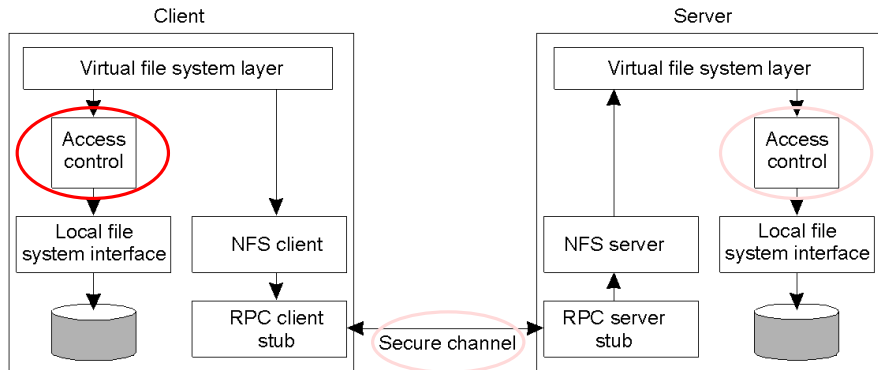


Abb. aus Tanenbaum/Steen

Sicherheit

Prinzipielle Architektur

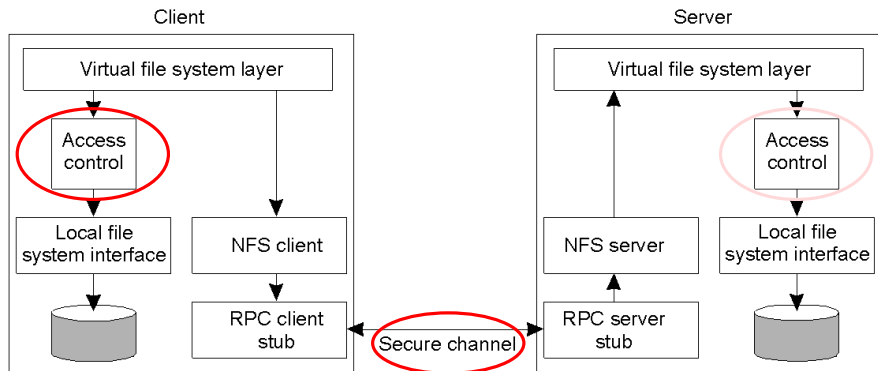


Abb. aus Tanenbaum/Steen

Sicherheit



Prinzipielle Architektur

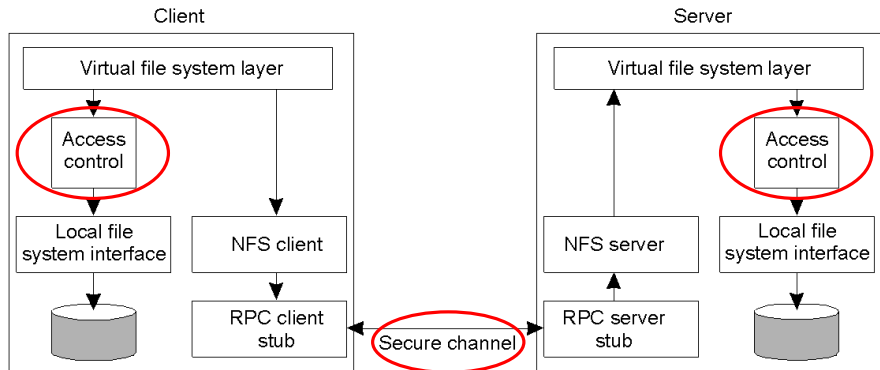


Abb. aus Tanenbaum/Steen

Sichere RPCs



in Version 3:

- Nur Authentifizierung
 - ▶ „System“
 - ★ basierend auf UNIX (uid, gid)
 - ★ übertragen im Klartext ohne Signatur (Server traut Client)
 - ▶ Diffie-Hellman
 - ★ selten genutzt
 - ★ wegen zu kurzer Schlüssel heute als definitiv unsicher eingestuft
 - ▶ Kerberos

in Version 4:

- Keine eigenen, fest eingebauten Mechanismen
- Unterstützung von RPCSEC_GSS
 - ▶ Sicherheitsumgebung für verschiedenste einklinkbare Mechanismen
 - ▶ Neben Authentifizierung auch Integrität und Vertraulichkeit

Sichere RPCs (2)

Architektur sicherer RPCs in Version 4:

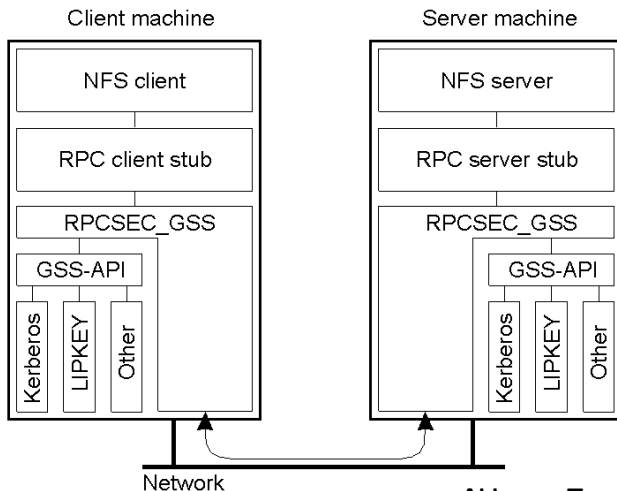


Abb. aus Tanenbaum/Steen

Zugriffskontrolle



in Version 3:

- UNIX-Rechteüberprüfung (uid, gid) auf Server-Seite

in Version 4:

- ACL-basiert
- Subjekte stark differenziert

Zugriffskontrolle (Operationen)



| Operation | Beschreibung |
|-------------------|---|
| 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 |

Zugriffskontrolle (Subjekte)



| Benutzertyp | Beschreibung |
|---------------|--|
| 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 |

Nach Tanenbaum/Steen

AFS und Coda



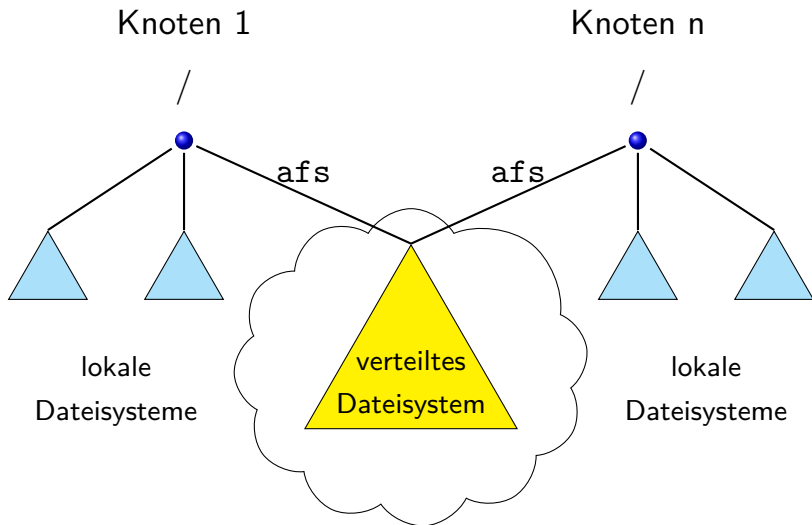
Andrew File System (AFS)

- CMU zusammen mit IBM, 1983-1989, danach Transarc
- Dateisystem für den Campus mit > 5.000 aktiven Studenten
- Ziele
 - ▶ Ortstransparenter, globaler, gemeinsam genutzter Dateinamensraum, erreichbar über den lokalen Namen /afs
 - ▶ Hohe Performance
 - ▶ Hohe Verfügbarkeit
 - ★ Replikation
 - ▶ Hohe Sicherheit
 - ★ Gesicherte Authentifikation
 - ★ Verschlüsselte Übertragung
 - ★ ACLs zur Zugriffskontrolle
 - ▶ Automatische Migration von Home-Verzeichnissen von Benutzern

Coda

- Weiterentwicklung von AFS-2

Dateinamensraum



Gesamtarchitektur

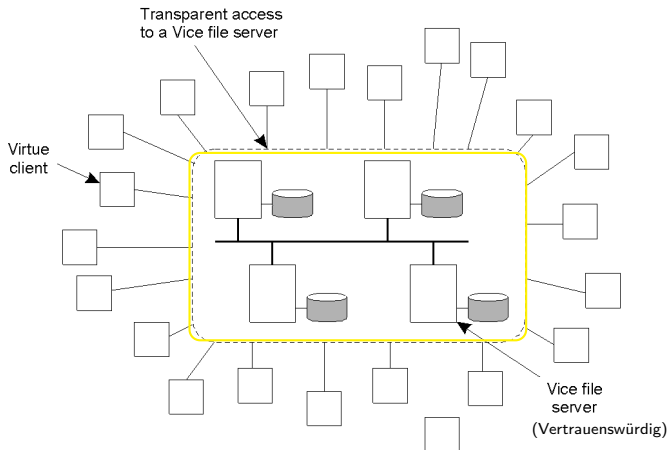
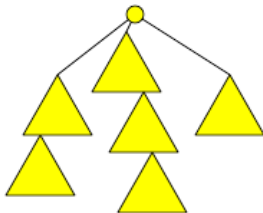


Abb. aus Tanenbaum/Steen

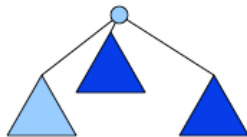
Volumes



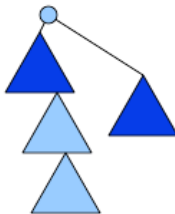
**Mountbare
Volumes**



**Struktur
des globalen Dateiraums
in Vice**



Knoten 1



Knoten n

**reale Struktur
in den Knoten
(Teilgraph)**



repliziert

Architektur eines Virtue-Clients

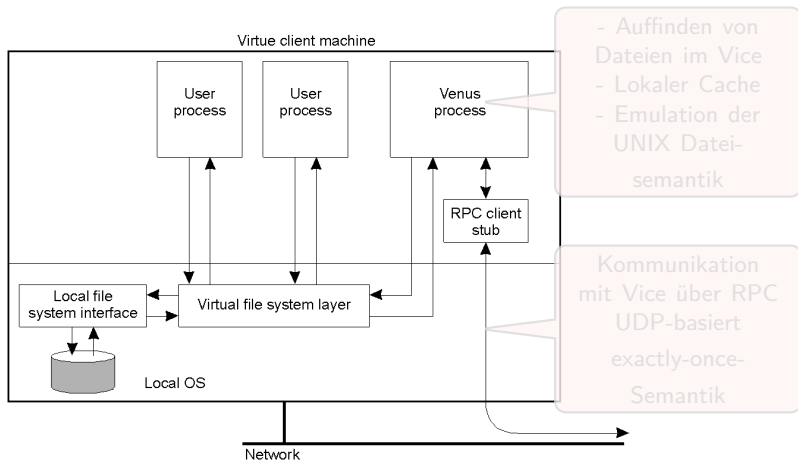


Abb. aus Tanenbaum/Steen

Architektur eines Virtue-Clients

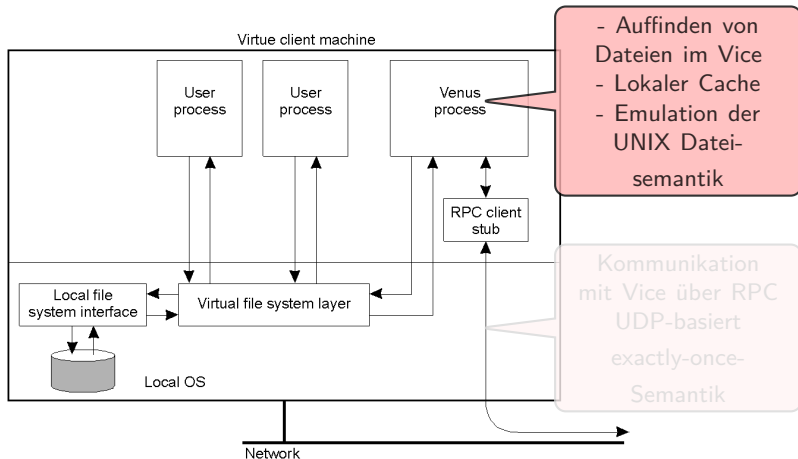


Abb. aus Tanenbaum/Steen

Architektur eines Virtue-Clients

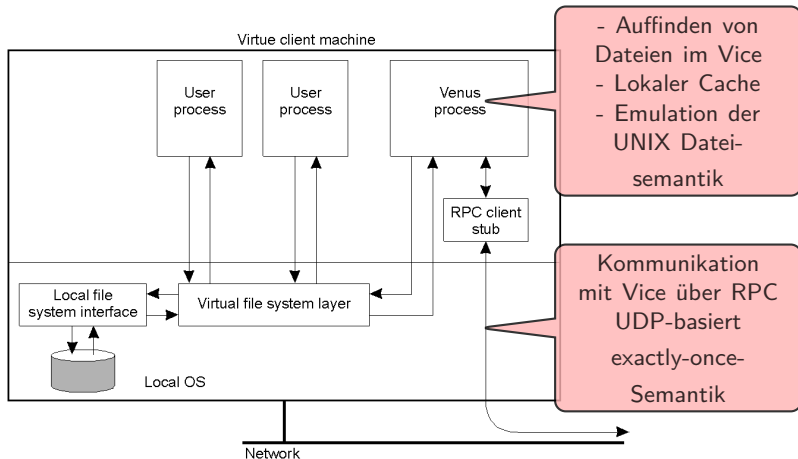


Abb. aus Tanenbaum/Steen

Eigenschaften



- gemeinsam genutzte Dateien mit Sitzungssemantik
- Lokales Caching von ganzen Dateien bis AFS-2, von groSSen Dateiblöcken (64 kB) ab AFS-3
- Cache-Kohärenz
 - ▶ Überprüfen der Gültigkeit des Caches nicht bei jedem `open()` notwendig
 - ▶ Callback-Verfahren, d.h. explizites Invalidieren durch den Server, bevor ein anderer Client Schreibrecht bekommt.

Sicherheit



- Organisation
 - ▶ Vice-Server sind vertrauenswürdig
 - ▶ keine Benutzer-Anwendungen auf Servern
 - ▶ Einführung administrativer Zellen zur Erhöhung der Skalierbarkeit
- Subjekte
 - ▶ Benutzer
 - ▶ Gruppen
- Authentifizierung
 - ▶ spezielle Authentication Server, Kerberos (ab AFS-3)
 - ▶ sicherer RPC
- Zugriffskontrolle
 - ▶ ACLs für Verzeichnisse definiert, gelten für alle Dateien des Verzeichnisses

Coda



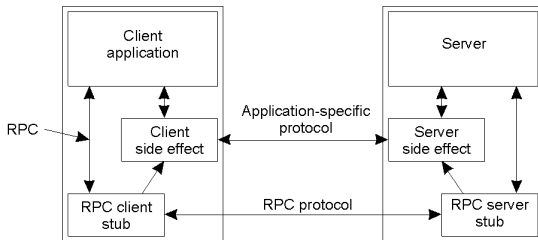
- Weiterentwicklung von AFS-2 ab 1987
- Ziele:
 - ▶ Hohe Verfügbarkeit der Daten
 - ▶ Client soll weiterarbeiten können, auch wenn Server vorübergehend nicht erreichbar ist (Netzpartitionierung)
 - ▶ Einbeziehung von mobilen Rechnern (gewollte Netzpartitionierung)

Kommunikation



RPC2-System

- Nutzung von internem Multithreading für Venus und Vice
- Unterstützung von sog. Nebeneffekten



- MultiRPCs (transparenter Aufruf mehrerer Server) genutzt für Cache-Invalidierung (Parallele Einzelaufrufe oder Nutzung von IP-Multicast)

File IDs

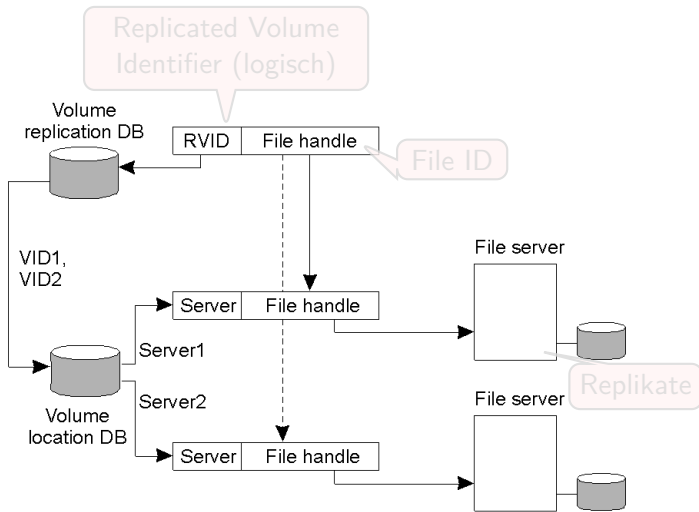


Abb. aus Tanenbaum/Steen

File IDs

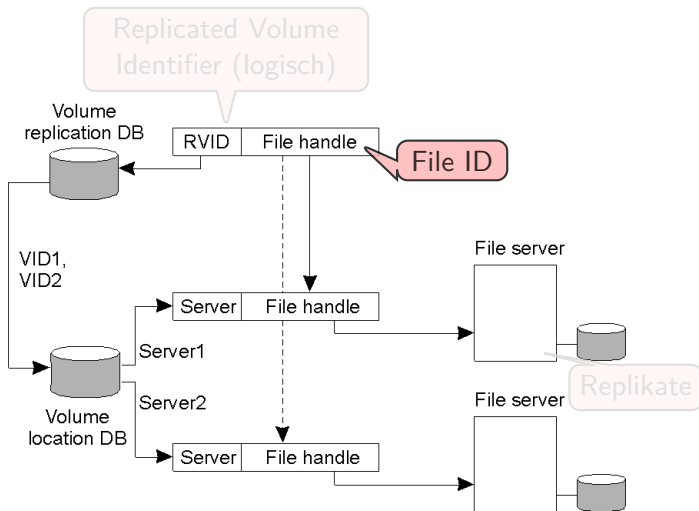


Abb. aus Tanenbaum/Steen

File IDs

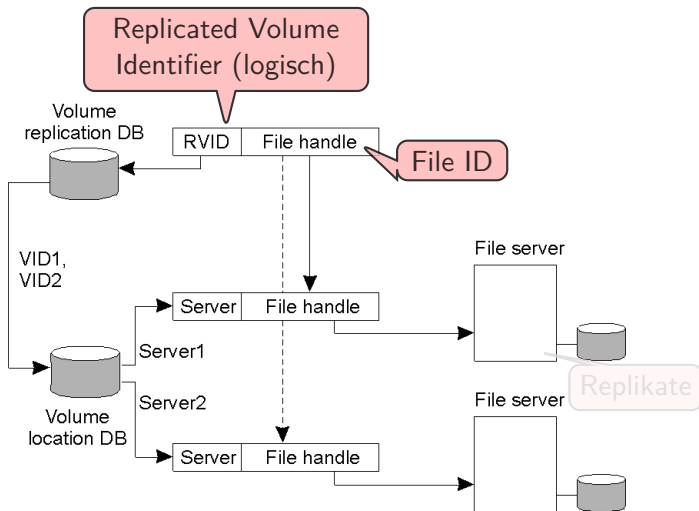


Abb. aus Tanenbaum/Steen

File IDs

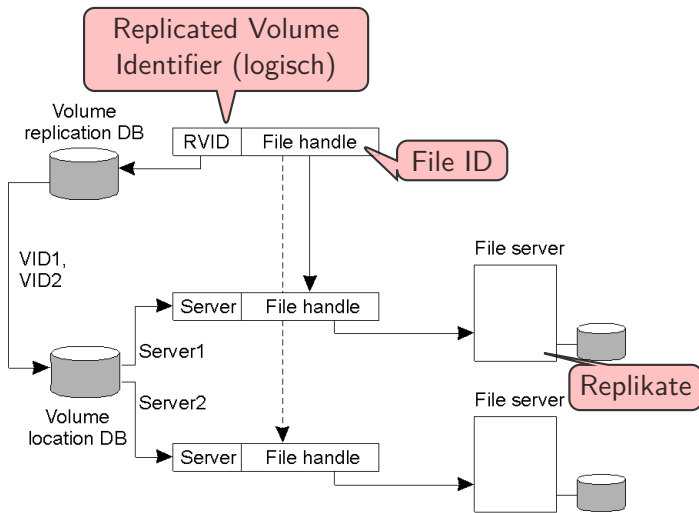


Abb. aus Tanenbaum/Steen

Eigenschaften im Normalbetrieb

- Sitzungssemantik
- ein Schreiber oder mehrere Leser
- „Transaktions“-Verhalten

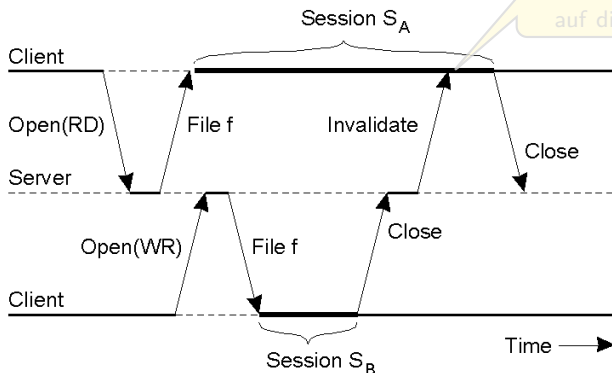


Abb. aus Tanenbaum/Steen

Eigenschaften im Normalbetrieb

- Sitzungssemantik
- ein Schreiber oder mehrere Leser
- „Transaktions“-Verhalten

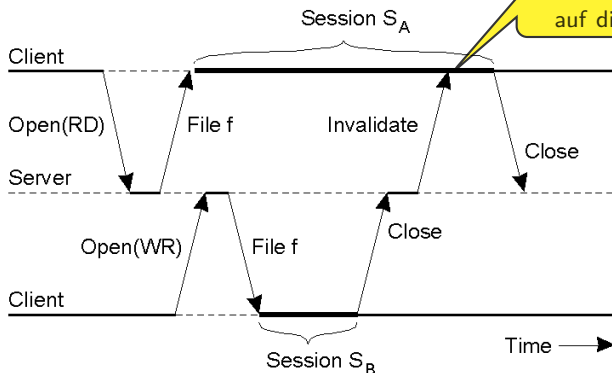


Abb. aus Tanenbaum/Stein

Caching



- Bei `open()` wird Datei in Client-Cache geladen
- Server gibt Callback-Versprechen
- Zur Invalidierung schickt Server Callback Break

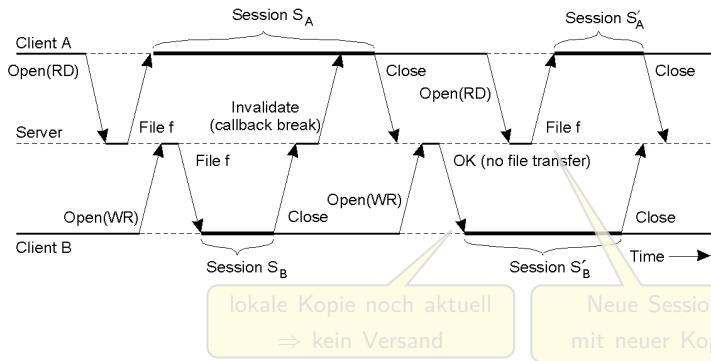


Abb. aus Tanenbaum/Steen

Caching



- Bei `open()` wird Datei in Client-Cache geladen
- Server gibt Callback-Versprechen
- Zur Invalidierung schickt Server Callback Break

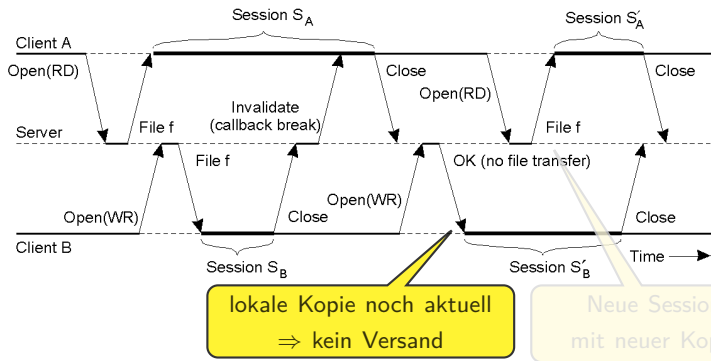


Abb. aus Tanenbaum/Steen

Caching



- Bei `open()` wird Datei in Client-Cache geladen
- Server gibt Callback-Versprechen
- Zur Invalidierung schickt Server Callback Break

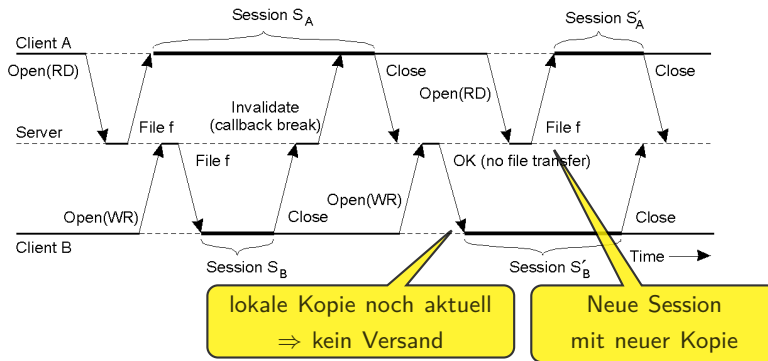


Abb. aus Tanenbaum/Steen

Server-Replikation und Netz-Partitionierung



- Volume ist Einheit der Replikation
- Volume Storage Group (VSG)
 - ▶ Menge der Server mit Kopie eines Volumes
- Accessible Volume Storage Group (AVSG)
 - ▶ Teilmenge von VSG, die Client kontaktieren kann
- Lesen von einem Replikat, Schreiben an alle mittels MultiRPC
- Optimistische Strategie für Dateireplikation
 - ▶ Bei Partitionierung dürfen mehrere Schreiber existieren und auf ihre jeweilige AVSG zurückschreiben
- Führen von Versionsvektoren entsprechend Vektorzeitstempel (vgl. Kap. 7) und Überprüfung bei Aktualisierung
- Spätere Zusammenführung verschiedener Versionen erfordert z.T. manuelle Hilfe

Verbindungsloser Betrieb



- Verbindungslos: AVSG = \emptyset , dann Nutzung der lokalen Kopie
- Konflikterkennung bei Übertragung auf den Server (s.o.)
- Beobachtung
 - ▶ Konflikte selten, da selten eine Datei von mehreren Prozessen gleichzeitig modifiziert wird
- Problem
 - ▶ Relevante Dateien im lokalen Cache haben, wenn Verbindungslosigkeit eintritt
- Ansatz: Hoarding (Horten von Dateien)
 - ▶ Heuristisches Verfahren
 - ▶ Explizite Angabe von Dateien und Verzeichnissen durch Benutzer
 - ▶ Priorisierung durch Abgleich mit aktueller Zugriffsinformation
 - ▶ Cache-Ausgleich (Hoard-Walk) alle 10 min
 - ▶ Gute Erfahrungen, aber es kommt natürlich vor, dass gelegentlich relevante Dateien fehlen

Zusammenfassung



| File-associated data | Read-lock | Write-Lock |
|-----------------------------|---------------------|-------------------------|
| 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 |

Nach Tanenbaum/Steen

Speichernetze



Nutzung von Netzwerktechnologie und verteilten Systemen innerhalb von Speichersystemen

Motivation

- Kostenreduktion
 - ▶ Geringere bereitgestellte Speicherkapazitäten
 - ▶ Zentrale Administration
- Höhere Flexibilität in der Zuordnung
 - ▶ Schnelle Anpassbarkeit an neue Bedürfnisse
- Skalierbarkeit
 - ▶ Kleine bis sehr groSSe Speicherkapazitäten
- Möglichkeit für Disaster Recovery
 - ▶ Datenspiegelung an entfernte Stellen

Architekturansätze



Wesentliche heutige Architekturansätze

- Direct Attached Storage (DAS) (klassischer lokaler Speicher)
- Storage Area Networks (SAN)
- Network-Attached Storage (NAS)
- Content Adressed Storage (CAS)

Im Folgenden vorgestellt

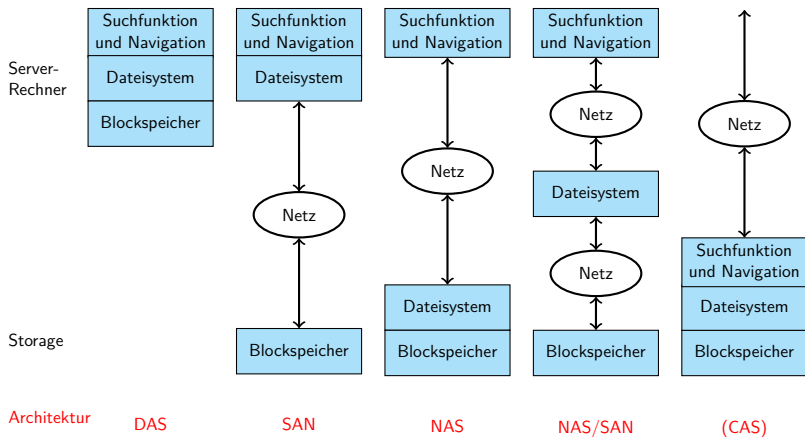
Speichersysteme als geschichtete Systeme



Grundlegende Aufteilung der Funktionalität von Informationsspeicherung

| | |
|-----------------------------|--|
| Suchfunktion und Navigation | Pfadnamen, Query, Index, Metadaten, ... |
| Dateisystem | Abbildung auf logische Blockmenge, z.B. NFS, AFS, Microsoft SMB/CIFS |
| Blockspeicher | Abbildung auf phys. Speichergerät |

Überblick zur Integration von Netzen



DAS: Direct Attached Storage



- Traditionelle, lokal angeschlossene Speichergeräte
 - Lokales Betriebssystem enthält Dateisystem und Gerätetreiber
 - Typische Geräteschnittstellen
 - ▶ IDE/ATA, SCSI, Serial ATA (SATA), ...
 - Beschränkungen
 - ▶ Anzahl vorhandener Kanäle
 - ▶ Anzahl anschließbarer Devices je Kanal
 - ▶ maximale Entfernung ca. 1-25 m
- ⇒ kein Disaster-Recovery möglich
- ▶ Performance
- ⇒ schlechte Skalierbarkeit

SAN: Storage Area Network

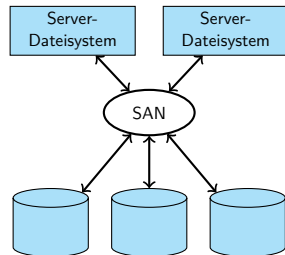


Funktionsweise

- SAN bietet Blockspeicher (z.B. Festplatten als Logical Devices)
- Server-Betriebssysteme enthalten ein oder mehrere Dateisysteme
- Blockspeicher wird über das SAN von Servern zugegriffen

Vorteile

- Sehr leichte Erweiterbarkeit
- Sehr flexible Zuordenbarkeit
- Hohe Skalierbarkeit
- Basis für Replikation, auch für Disaster Recovery
- Bootbare Netzwerkpartitionen
- Besonders geeignet für Anwendungen, die mit Volumes (ohne Dateissystem) arbeiten, z.B. DBMSe



SAN: Storage Area Network(2)



Verbreitete Netzwerke

- Fibre Channel (FC)
 - ▶ Dediziertes Netzwerk mit 1/2/4/8/16 GBit/s über Glasfaser
 - ▶ geschichtetes Protokoll FC-0 bis FC-4
 - ▶ verschiedene Übertragungsprotokolle in FC-4 einbettbar, typisch: seriell SCSI-3
 - ▶ Skalierbarkeit über Switches
 - ▶ Ausdehnung bis 10-100 km
- Internet SCSI (iSCSI)
 - ▶ neuerer Standard (RFC 3720, 2004)
 - ▶ Nutzung von preiswerter Internet-Technologie zur Blockübertragung
 - ▶ TCP/IP als unterlagertes Protokoll
 - ▶ Ausdehnung beliebig
 - ▶ Security- und QoS-Standards nutzbar
 - ▶ hohes Wachstum mit Gbit/s-Ethernet erwartet

NAS: Network Attached Storage

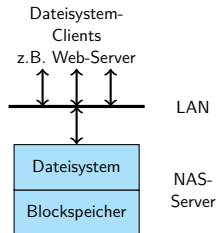


Funktionsweise

- NAS bietet Netzwerkdateisysteme (z.B. NFS, SMB/CIFS (Server Message Block bzw. Nachfolger Common Internet File System))
- Clients können Dateisysteme nutzen

Vorteile

- Speicher-Konsolidierung
- Erweiterbarkeit
- Skalierbarkeit
- Managebarkeit
- Besonders geeignet für Anwendungen, die auf Dateizugriffen basieren, z.B. Web-Anwendungen, Home-Verzeichnisse



NAS: Network Attached Storage (2)



Beispiele:

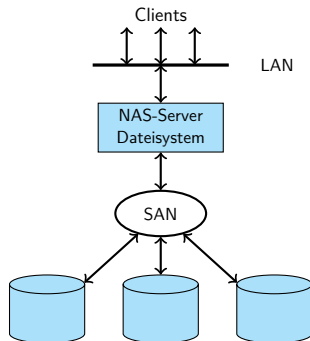
- Home Server
 - ▶ Synology
 - ▶ Zyxel
- IBM Storvize V7000

NAS / SAN Verbund



Funktionsweise

- NAS und SAN können kombiniert eingesetzt werden
- Vorteile lassen sich so kombinieren



CAS: Content Adressable Storage



Grundlage des Ansatzes: Unveränderliche Information

- daher auch Fixed Content Storage (FCS) genannt
- Ziel: Archiv-Speicher

Teilproblem des Information Lifecycle Management (ILM)

- Massendaten (Hunderte von Terabytes bis Petabytes)
- Langlebigkeit
- Integrität
- Unveränderbarkeit von Dokumenten z.T. gesetzlich gefordert

Anwendung im Bereich Content Management

- Digitale Medien (Ton-, Bild-Dokumente)
- email-Archivierung
- Gesundheitswesen (Röntgenbilder etc.)
- ...

CAS: Content Adressable Storage (2)



Funktionsweise

- Content Identifier als Referenz
- Bestimmung eines Content-Identifiers
 - ▶ ausschliesslich aus Inhalt (analog Hash-Wert) \Rightarrow ortsunabhängig
 - ▶ oder aus Speicherort (invertiert)
- System ermittelt aus Content Identifier Location für Zugriff

Beispiele

- Erstes System: EMC Centera 2002
- iTernity iCAS (Software-Lösung)
- Versch. Open Source Lösungen, z.B. Keep Content Adressable Storage

Zusammenfassung



- Verteilte Dateisysteme ermöglichen den nebenläufigen Zugriff unterschiedlicher Nutzer auf Dateien unabhängig von ihrem Speicherort.
- Der Zugriff auf Dateien kann je nach Konsistenz-Semantik zu unterschiedlichen Ergebnissen führen.
- Durch Aufteilung der Funktionalität zur Informationsspeicherung auf unterschiedliche Systeme im Netzwerk lassen sich u.a. Skalierbarkeit, Fehlertoleranz und Erweiterbarkeit verbessern.