Verteilte Systeme

15. Dateisysteme

Motivation

- Speicherung persistenter Daten
- Vorteile durch Verteilung
  - Ortsunabhängiger Zugriff auf Dateien
  - Erhöhte Zuverlässigkeit
  - Verbesserte Leistung
- Historisch mit die ersten verteilten Anwendungen
- Netzwerkdateisysteme
  - Primär ortsunabhängiger Zugriff
- Verteilte Dateisysteme
  - Erhöhte Zuverlässigkeit durch Replikation
  - Erhöhter Durchsatz
    - Schnelle Netze

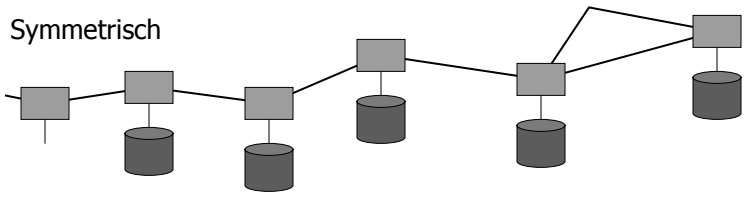
Transparenz

- Ziel: Single-System-Image
- Verteilung soll für Anwendungen möglichst transparent sein
- Transparenzbegriffe
  - Zugangstransparenz
    Identische Programmierschnittstelle
  - Ortstransparenz
    Der Dateiort (lokal, entfernt) ist nicht ermittelbar
  - Nametransparenz
    Der Dateiname ist überall gleich
  - Migrationstransparenz
    Der Name bleibt auch bei einer Dateimigration gleich
  - Leistungstransparenz
    Die Geschwindigkeit beim lokalen und entfernten Zugriff ist gleich
  - Fehlertransparenz
    Fehler aufgrund der verteilten Systemstruktur werden maskiert

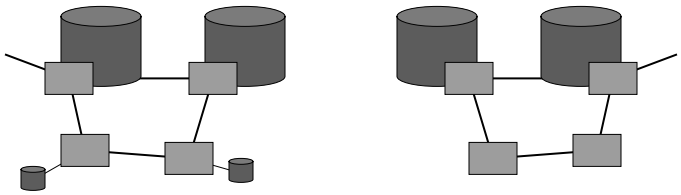
Verteilte Systeme, Wintersemester 2000/2001 Folie 15.3

Architekturvarianten

- Symmetrisch



- Asymmetrisch



Verteilte Systeme, Wintersemester 2000/2001 Folie 15.4

Nutzungscharakteristik

- Die meisten Dateien sind klein (<10 KB)
  - Caching ganzer Dateien sinnvoll
- Dateien werden häufiger gelesen als modifiziert
  - Caching effektiv
- Dateien haben häufig eine kurze Lebensdauer
  - Temporäre Dateien
- Datei-Sharing ist selten
  - Caching auf Clientseite, geringe Konfliktwahrscheinlichkeit
- Prozesse nutzen nur selten mehrere Dateien gleichzeitig
- Hohe Wahrscheinlichkeit für Folgezugriffe
  - Prefetching
- Sequentieller Zugriff überwiegt

Sharing von Dateien

- Nutzung einer Datei durch mehrere Prozesse
- Konfliktgefahr
- Ansätze
  - Unix-Semantik
    - Änderungen werden sofort sichtbar
  - Session-Semantik
    - Änderungen werden erst nach dem Schließen sichtbar
  - Immutable-File-Semantik
    - Unveränderbare Dateien
    - Änderung erzeugt neue Version
  - Transaktionssemantik
    - Dateizugriff über Transaktionen

Caching

- Wesentlich für Leistungssteigerung (Leistungstransparenz)
- Granularität
  - Ganze Dateien
  - Größere Portionen (inkl. Read-Ahead)
  - Seiten
- Ort für Caching
  - Serverseitig
    - Reduziert Plattenzugriffe
    - Netzverkehr bleibt
  - Clientseitig
    - Reduziert Netzverkehr
    - Verbessert Zuverlässigkeit (Replikate)
    - „Disconnected Operation" (Mobile Systeme)
    - Replikatkonsistenz sicherstellen

Wahrung der Cache-Kohärenz

- Zeitpunkt der Aktualisierung bei Replikaten
- Write-Through
  - Änderung sofort zurückschreiben
  - Keine Reduktion des Netzverkehrs beim Schreiben
  - Nach jedem Byte zurückschreiben?
- Delayed-Write
  - Größeres Konfliktfenster
- Write-On-Close
  - Ideal bei Session-Semantik
  - Sonst noch größeres Konfliktfenster
- Zentrale Kontrolle
  - Deadlockgefahr
  - Problematik Skalierbarkeit

Zustandslose oder zustandsbehaftete Server

Zustandslos

- Fehlertoleranz verhältnismäßig einfach realisierbar
- Öffnen und Schließen von Dateien unnötig
- Anzahl offener Dateien unbeschränkt
- Keine Probleme mit Client-Abstürzen (Verwaister Zustand)
- Längere Nachrichten
- Höhere Latenz

Zustandsbehaftet

- Fehlertoleranz aufwendig
- Read-Ahead möglich
- Idempotenz von Operationen leichter erreichbar
- Sperren von Dateien möglich
- Kürzere Nachrichten
- Höhere Leistung

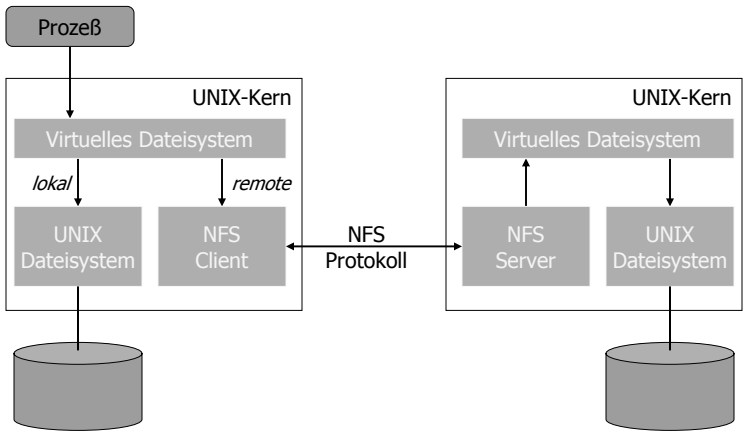
Verteilte Systeme, Wintersemester 2000/2001 Folie 15.9

15.1 Netzwerkdateisystem

Beispiel: NFS

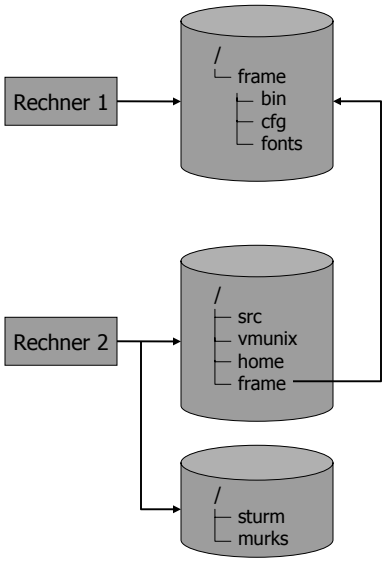
- Erste Realisierung 1985
  - Basiert auf SUN-RPC, Frei zugänglich, Quasi-Standard
- Ziele
  - Heterogene Systeme
  - Hohe Portabilität
- Zugangstransparenz
- Ortstransparenz ist möglich
- Fehlertransparenz
  - NFS-Server sind zustandslos
  - Operationen sind idempotent
- Leistungstransparenz
  - Caching auf Client- und Serverseite
  - Ca. 20% Overhead gegenüber einem lokalen Zugriff

NFS-Architektur



Mouting

- Filesystem
  - In sich abgeschlossener Dateibaum mit Wurzel
  - Paßt auf einen Datenträger
- Rechner exportieren Filesysteme
  - Wer darf darauf zugreifen?
  - Zugriffsrecht?
- Spezieller Mountserver
  - mountd (statisch)
  - Automounter (dynamisch)
  - Ebenfalls RPC-Protokolle



Verteilte Systeme, Wintersemester 2000/2001 Folie 15.13

```
program NFS_PROGRAM {
    version NFS_V3 {

        void NFSPROC3_NULL(void) = 0;
        ... NFSPROC3_GETATTR(...) = 1;
        ... NFSPROC3_SETATTR(...) = 2;
        ... NFSPROC3_LOOKUP(...) = 3;
        ... NFSPROC3_ACCESS(...) = 4;
        ... NFSPROC3_READLINK(...) = 5;
        READ3res NFSPROC3_READ(READ3args) = 6;
        ... NFSPROC3_WRITE(...) = 7;
        ... NFSPROC3_CREATE(...) = 8;
        ... NFSPROC3_MKDIR(...) = 9;
        ... NFSPROC3_SYMLINK(...) = 10;
        ... NFSPROC3_MKNOD(...) = 11;
        ... NFSPROC3_REMOVE(...) = 12;
        ... NFSPROC3_RMDIR(...) = 13;
        ... NFSPROC3_RENAME(...) = 14;
        ... NFSPROC3_LINK(...) = 15;
        ... NFSPROC3_READDIR(...) = 16;
        ... NFSPROC3_READDIRPLUS(...) = 17;
        ... NFSPROC3_FSSTAT(...) = 18;
        ... NFSPROC3_FSINFO(...) = 19;
        ... NFSPROC3_PATHCONF(...) = 20;
        ... NFSPROC3_COMMIT(...) = 21;
    } = 3;
} = 100003;
```

Verteilte Systeme, Wintersemester 2000/2001 Folie 15.14

Beispiel **NFSPROC3_READ**

- Argumente

```
struct READ3args {
    nfs_fh3 file;
    offset3 offset;
    count3 count;
};
```

  - **file**: Maximal 64 Byte Dateideskriptor (Handle)
  - **offset**: 64 Bit Integer
  - **count**: 32 Bit Integer
- Dateideskriptor?

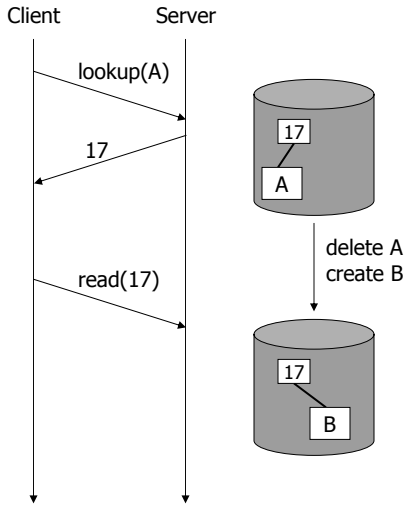
- Resultat

```
struct READ3resok {
    post_op_attr file_attr;
    count3 count;
    bool eof;
    opaque data<>;
};

struct READ3resfail {
    post_op_attr file_attr;
};

union READ3res
switch (nfsstat3 status) {
    case NFS3_OK:
        READ3resok resok;
    default:
        READ3resfail resfail;
};
```

Verteilte Systeme, Wintersemester 2000/2001 Folie 15.15

Zustandsloser Server und Dateideskriptor?

- Deskriptor darf nicht vom Serverzustand abhängen
- Knotenweit eindeutige und für die Lebensdauer der Datei gültige Deskriptoren möglich
- I-Nodes
- Deskriptor = 3-Tupel
  - Identifikation des Filesystems (Welche Platte?)
  - I-Node
  - Generationsnummer der I-Node

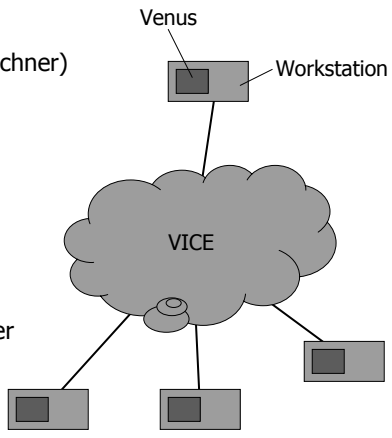


Verteilte Systeme, Wintersemester 2000/2001 Folie 15.16

15.2 AFS und Coda

Andrew File System

- CMU in Kooperation mit IBM (1983-)
  - Satyanarayanan
- Ziele
  - Skalierbarkeit (mehr als 5000 Rechner)
  - Effizienz
  - Schutz
- Aufbau
  - VENUS - Clients
    - Auffinden der Dateien
    - Lokaler Cache
    - UNIX-Semantik
  - VICE - Vertrausenswürdige Server
- Identischer Teilbaum /afs auf jeder Workstation



Verteilte Systeme, Wintersemester 2000/2001 Folie 15.18

Chronologie

- AFS-1 (1984)
  - Struktur von /afs auf den Servern gespiegelt
  - Multi-Process Server
  - Auflösung der Pfadnamen in VICE
  - Cache-Kohärenz: Lokale Kopie letzte Version?
    - Anfrage beim Server
- AFS-2 (1986)
  - Server informiert über Cache-Invalidierungen (Callback)
  - Client-Caching von Directories und lokale Pfadauflösung
  - Multi-Threaded Server (nicht-preemptive Threads)
  - Unterstützung von Nurlese-Replikation
- AFS-3 (1989)
  - Administrative Einheiten (Cells)
  - Kooperation zwischen Zellen
  - Caching von Dateiteilen (weniger als 64 KB)

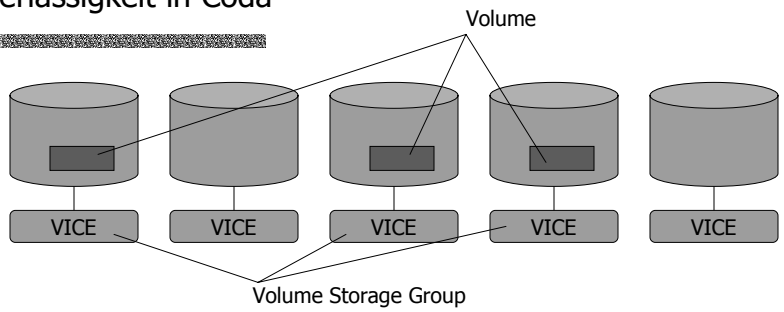
Schutz

- VICE-Server sind vertrauenswürdig
  - Physischer Schutz
  - Keine Benutzerprozesse
- Schutzbereiche (Protection Domains)
  - Benutzer
  - Gruppe = Menge von Benutzern und anderen Gruppen
- Group Inheritance
  - Kumulative Privilegien eines Benutzers
- Protection Server
- Authentisierung
  - Seit AFS-3 Kerberos mit geheimen Schlüsseln
- Zugriffskontrollisten
  - Negative Rechte möglich
  - Owner-Schutzbits (vgl. UNIX)

Coda

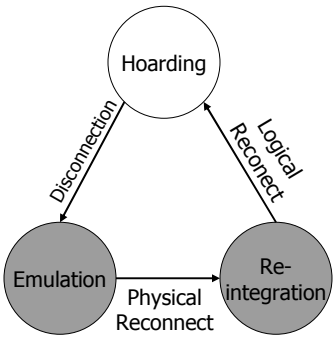
- CMU, Satayanarayanan, seitz 1990
- Ziele
  - Ständige Verfügbarkeit
    - Kompensation von Serverausfällen
    - Kompensation von Netzwerkausfällen (Partitionierung)
  - Integration portabler Computer
    - Gewollte Netzpartitionierung
- Nachfolgesystem von AFS-2
  - Caching ganzer Dateien (inhärente Kompensation von Ausfällen)
  - Cache-Koheränz durch Callbacks
  - Dynamische Dateilokalisierung

Zuverlässigkeit in Coda



- Replikation bestimmter Teile des gemeinsamen Dateisystems
- Volume Storage Group
- Venus speicher „Accessible VSG" (AVSG)
  - Periodische Tests aktualisieren AVSG
  - Verkleinern bei Ausfall eines Servers
  - Vergrößern bei „Wiedereintritt" eines Ersatzservers
  - Preferred Server (PS) innerhalb einer AVSG

Disconnected Operation

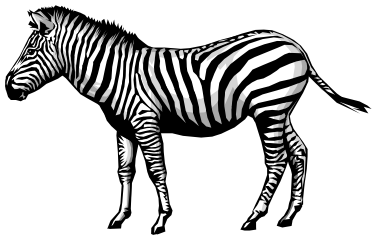


- AVSG = ∅
- Optimistisches Verfahren
  - Venus arbeitet auf den Dateien im Cache
- Hoarding
  - Soviel wie möglich lokal cachen
  - „Hoard Walking": Was bleibt im Cache?
  - Venus beobachtet Zugriffe
  - „Hoard Profiles"
  - „Hoard Databases" (HDB)
- Emulation
  - Venus übernimmt Serverrolle
  - Reply-Log aufnehmen
  - Optional blockieren
  - Kritische Daten werden in einem „Recoverable Virtual Memory" gesichert (Transaktionen)
- Reintegration
  - Reply-Log an AVSG senden und verarbeiten
  - Viele Konflikte automatisch lösbar
  - Rest manuell mit Werkzeugunterstützung

15.3 Logbasierte Dateisysteme

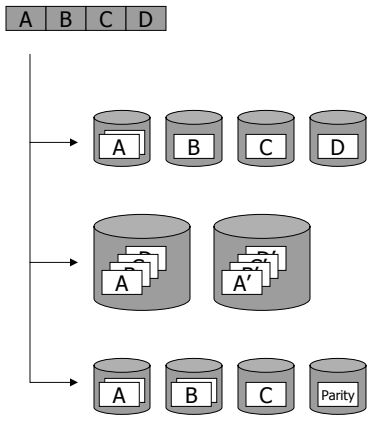
Zebra Network File System

- J.H. Hartman, J.K. Ousterhout
  University of Arizona, SUN
- „Network RAID"
  - Kummulativer Durchsatz
    - Leistung eines einzelnen Servers inkl. E/A-Bandbreite nicht mehr der Flaschenhals
    - Voraussetzung: Hohe Netzbandbreiten zwischen jeweils zwei Rechnern
  - Hohe Verfügbarkeit
- Zentrale Ansätze
  - Striping von Dateien
  - Append-only Log bei Änderungen (Kein Update-in-Place)



Verteilte Systeme, Wintersemester 2000/2001 Folie 15.25

Rückblick: RAID

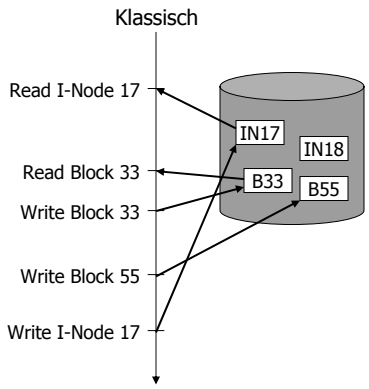


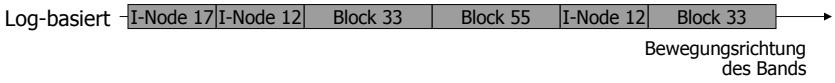
- Redundant Arrays of Independent Disks
- 6 Stufen
  - RAID 0: Reines Stripping
  - RAID 1: Spiegelung
  - RAID 2: Striping + ECC
  - RAID 3: Striping + Parity
  - RAID 4: Große Stripes + Parity
  - RAID 5: Große Stripes + Rotating Parity
- Gängig sind RAID 3 und RAID 5
- Große Writes bevorzugt
  - Overhead 1/(N-1) bei N Platten
- Kleine Writes teuer
  - Datenstripe laden
  - Datenstripe schreiben
  - Parityblock laden
  - Neuen Parityblock schreiben

Verteilte Systeme, Wintersemester 2000/2001 Folie 15.26

Rückblick: Log-based File Systems (LFS)

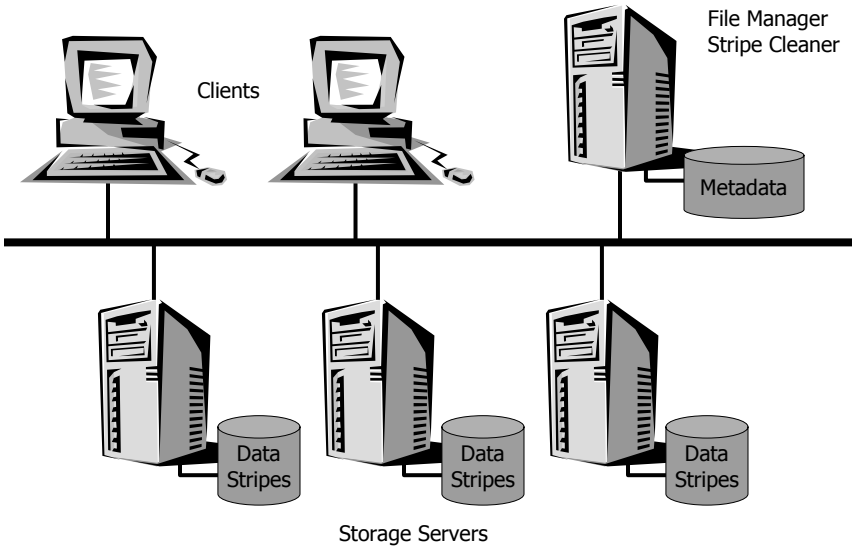
- M. Rosenblum, J.K. Ousterhout (SUN)
- Platte wird als Append-Only Log verwendet
  - Anfügen neuer/geänderter Datenblöcke
  - Anfügen neuer/geänderter Metadaten
- Effizientes Schreiben kleiner Dateien
  - Kleine Dateien sind häufig
  - Gemessen wurde z.B. Faktor 10
- Probleme
  - Wiederfinden der aktuellsten Daten
  - Speicherverwaltung





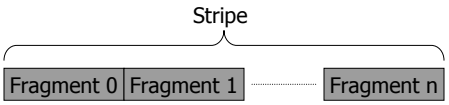
Verteilte Systeme, Wintersemester 2000/2001 Folie 15.27

Zebra-Architektur



Verteilte Systeme, Wintersemester 2000/2001 Folie 15.28

Client



Lesender Zugriff

- Anfrage beim File Manager
  - Welche Stripe Fragmente liegen auf welchem Storage Server?
- Laden der benötigten Fragmente
  - Gleichzeitig Aufträge an mehrere Storage Server absenden
- Weitergabe der Daten an Applikation

Schreibender Zugriff

- Einsammeln der Änderungen im Client-Log
  - Enthalten nur Daten
  - Stripes haben konstante Länge (512 KB im Prototyp)
- Bei vollen Stripes Parity-Information generieren
- Stripe-Fragmente an die einzelnen Storage-Server senden
- Metadaten an File Manager

Storage Server

- Speichern lediglich Stripe-Fragmente
- Kennzeichnung jedes Fragments
  - Identifikation des Clients
  - Sequenznummer des Stripes (Client-lokal eindeutig)
  - Offset des Fragments innerhalb des Stripes
- 5 Funktionen
  - Neues Fragment mit Kennzeichnung auf Platte speichern
  - Daten an existierendes Fragment anfügen
    - Fragment inkrementell erzeugen
    - Z.B. bei `fsync()` oder `flush()` durch Anwendung
  - Fragment teilweise oder vollständig lesen
  - Fragment löschen
  - Identifiziere Fragment
    - Z.B. zuletzt geschriebenes Fragment eines Clients
    - Notwendig für Crash Recovery

File Manager

- Verwaltung aller Metadaten
  - Datei- und Schutzattribute
  - Informationen über die Blockverkettung der Datei
  - Verzeichnisse
  - Symbolische Links
- Im Prototyp einfaches Dateisystem (LFS)
  - Dateien speichern die Metadaten für Zebra
- Probleme
  - File Manager kann zum Flaschenhals werden
    - Öffnen und Schließen erfordert Kommunikation
    - Vorschlag: Client-seitiges Caching der Metadaten
  - Single Point of Failure

Stripe Cleaner

- Stripes veralten
  - Nachfolgende Änderungen invalidieren frühere Blöcke
- Stripe Cleaner wird auf einem Client ausgeführt
- Stripe Cleaner ist normaler Userprozeß
  - "Leere" Stripes suchen und freigeben
  - "Fast leere" Stripes
    - Gültige Blöcke im Client-Log aufnehmen
    - Client-Log wird als neuer Stripe wieder zurückgespeichert
- Wie erkennt der Cleaner freie Stripes?
- Race-Conditions zwischen Cleaning und Zugriff?

Blöcke und Deltas

Delta

File Identifier
File Version
Block Number
(Stripe, Offset) Alt
(Stripe, Offset) Neu

- Client-Log besteht aus Blöcken und Deltas
- Block
  - Datenblöcke der Dateien
- Delta
  - Information über Blockänderungen
  - File Identifier entspricht I-Node
  - Ordnen der Deltas aus verschiedenen Logs über Versionsnummer (nach Crash)
  - Alter und neuer Blockzeiger (Stripe, Offset)
  - Create Block: Alter Blockzeiger = 0
  - Delete Block: Neuer Blockzeiger = 0
- Update Deltas von den Clients
  - Blöcke erzeugen, ändern und löschen
- Cleaner Deltas
  - "Verschieben" gültiger Blöcke
- Reject Deltas vom File Manager
  - Auflösen von Konflikten zwischen Client und Cleaner

Schreiben auf Datei

- Änderungen in Client-Cache speichern
- Zurückschreiben in die Storage Server
  - Periodisch alle 30 Sekunden (vgl. Sync)
  - Änderungen erreichen Größe eines Stripes
  - Anwendung ruft `fsync()` auf
  - File Manager fordert auf
- Senden aller Blöcke enschließlich Parity wird durch Deltas im Log vermerkt
- Versionsnummer der betroffenen Dateien inkrementieren
- Asynchroner RPC zwischen Client und Storage Server
  - Parallele Aufträge an mehrere Server
- Stripe bei vorzeitigem Rausschreiben sukzessive auffüllen

Frequenz von `fsync()`

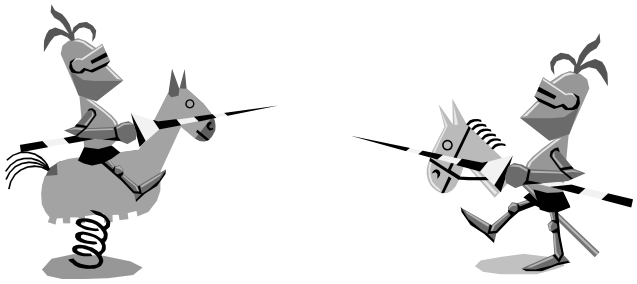
- Synchrones Rausschreiben der Blöcke auf die Storage Server
- Frequenz hat Auswirkungen auf Systemperformanz
- Transaktionsbasierte Last
  - 90% partielle Stripes
- Andere Last
  - <20% partielle Stripes

Der Stripe Cleaner im Detail

- Cleaner verarbeitet die Client-Logs
  - Deltas geben Aufschluß über frei gewordene Blöcke
  - Protokoll der Platzauslastung pro Stripe
  - Aufbau von Stripe Status Logs
    - Erleichert das Auffinden der Blöcke beim Kopieren
    - Kopie der zugehörigen Deltas
    - U.U. 1 Delta in zwei Stripe Status Logs
- Betroffene Dateien werden vom Cleaner nicht geöffnet
  - Race-Condition mit gleichzeitig zugreifenden Clients
- Optimistischer Ansatz
  - Cleaner Deltas
- Konfliktauflösung durch File Manager

Der Konflikt

- Bei Konflikt
  - Cleaner Delta: CD(X,*,17,(Stripe 12, 36644), New BP)
  - Update Delta: UP(X,*,17,(Stripe 33,46987), New BP)
- File Manager verwaltet Blockzeiger 17: (a,b)



Die Konfliktauflösung

- Bei Konflikt
  - Cleaner Delta: CD(X,*,17,(Stripe 12, 36644), New BP)
  - Update Delta: UP(X,*,17,(Stripe 33,46987), New BP)
- File Manager verwaltet Blockzeiger 17: (a,b)
- 1. Fall: Cleaner Delta kommt beim File Manager an
  - (a,b) == (12,36644): Kein Konflikt
  - (a,b) != (12,36644)
    - Update wurde bereits aufgenommen
    - Cleaner Delta ignorieren
    - Neuen Block in CD durch Reject Delta freigeben
- 2. Fall: Update Delta kommt an
  - (a,b) == (233,46987): Kein Konflikt
  - (a,b) != (233,46987)
    - Update nachziehen
    - Reject Delta für den neuen Block in CD

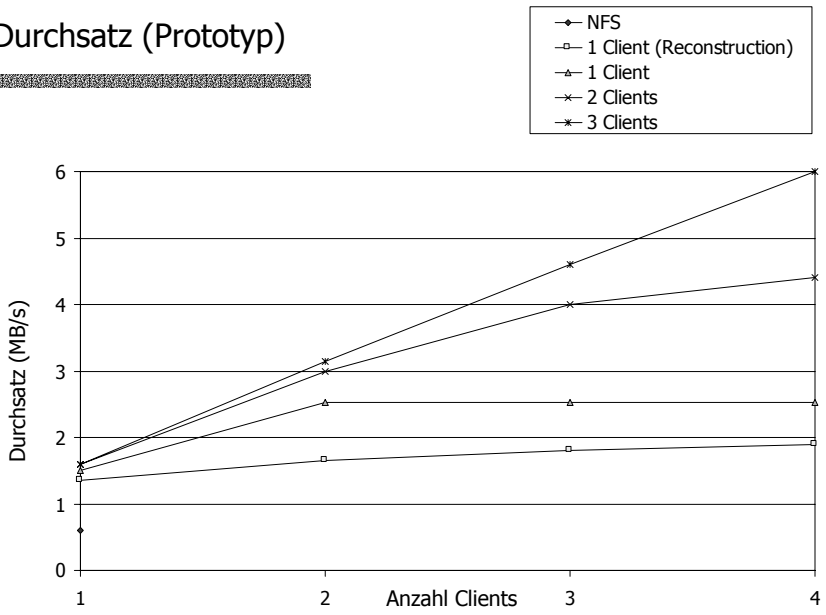
Lesen und gleichzeitiges Cleaning

- Client öffnet Datei und erhält Blockzeiger
- Client greift auf Block B zu
- Gleichzeitig kopiert Cleaner den Block in anderes Stripe
- 1. Fall: Stripe enthält noch gültige Blöcke
  - B weiterhin enthalten
  - Normaler Zugriff
- 2. Fall: Stripe wurde gelöscht
  - Storage Server meldet fehlerhaften Zugriff
  - Client fordert erneut Blockliste vom File Manager an

Weitere Fragen

- Hinzufügen eines Storage Servers?
- Bewußtes Entfernen eines Storage Servers?
- Unbewußtes Entfernen: Crash
  - Storage Server?
  - Stripe Cleaner?
  - File Manager?

Durchsatz (Prototyp)



Literatur

- J.H. Hartman, J.K. Ousterhout
  The Zebra Striped Network File System
  ACM TOCS, Vol. 13, No. 3, pp. 274-310, August 1995
- R. Sandberg, D. Goldberg, S. Kleinman, D. Walsh, B. Lyon
  Design and Implementation of the SUN Network File System
  Proc. Summer USENIX Conference, 1985
- M. Satyanarayanan
  Distributed File Systems
  in S. Mullender (Hrsg.), Distributed Systems
  pp. 353-383, Addison-Wesley, 2. Auflage, 1993
  (AFS und Coda)