

Skalierbarkeit in verteilten Systemen – Architekturen für verteilte Dateisysteme

Markus Brückner (vortrag@slash-me.net)

Hauptseminar Fachgebiet Verteilte Systeme und Betriebssysteme
TU Ilmenau

2004-06-02

Einführung

- Begriffsklärung

- Problemstellungen bei verteilten Dateisystemen

- Unterteilung verschiedener Dateisystemtypen

- Vorstellung Coda Filesystem

Ansätze zur Steigerung der Skalierbarkeit

- Replikation

- Caching

- Callbacks

- Hints

- Zentrale Nutzerverwaltung

- Gruppen/Rechtevererbung

- Negative Rechte

Fazit

Einführung

Begriffsklärung

Problemstellungen bei verteilten Dateisystemen

Unterteilung verschiedener Dateisystemtypen

Vorstellung Coda Filesystem

Ansätze zur Steigerung der Skalierbarkeit

Replikation

Caching

Callbacks

Hints

Zentrale Nutzerverwaltung

Gruppen/Rechtevererbung

Negative Rechte

Fazit

Was ist *Skalierbarkeit*?

Fähigkeit zur Handhabung wechselnder Lasten ohne grundsätzliche Änderung des Systems

Zwei Arten:

Last-Skalierbarkeit Konstantes Systemverhalten über große Lastbereiche hinweg

administrative Skalierbarkeit Möglichkeit zur effizienten Verwaltung großer und stark fluktuierender Subjekt- und Objektmengen

Verschiedene Aspekte des verteilten Systems benötigen Beachtung

- ▶ Verhalten bei steigender Systemlast
- ▶ Anpassung an verschiedene Bandbreiten des Kommunikationsnetzes
- ▶ Administrationsaufwand bei steigender Nutzerzahl
- ▶ Benutzerfreundlichkeit bei wachsender Größe des Systems

Skalierbarkeits-relevante Parameter

Nutzeranzahl steigende Nutzerzahlen führen sowohl zu erhöhter Last, als auch gesteigerten Administrationsaufwand

Datenmenge steigende Datenmengen erhöhen Last und Administrationsaufwand

Zugriffsverhalten Abhängig vom Nutzungsprofil (wenige langfristige parallele Zugriffe oder viele kurze Operationen) bieten sich unterschiedliche Strategien zur Lastverteilung an

verfügbare Netzbandbreite und -latenz Bei sinkender Bandbreite und/oder steigender Latenz werden Maßnahmen nötig, die die Performanz des Dateisystems erhalten

Unterteilung verschiedener Dateisystemtypen

Typ	Eigenschaften	Beispiel
Entferntes Dateisystem	<ul style="list-style-type: none"> ▶ 1:n Server-Client-Kommunikation ▶ enge Emulation der UNIX-Dateisystemsemantik 	NFS, Samba
Verteiltes Dateisystem	<ul style="list-style-type: none"> ▶ m:n Server-Client-Kommunikation ▶ meist lockerere Emulation der UNIX-Dateisystemsemantik 	Coda, AFS, Intermezzo
Peer-to-Peer System	<ul style="list-style-type: none"> ▶ keine Trennung Server\leftrightarrowClient 	Locus

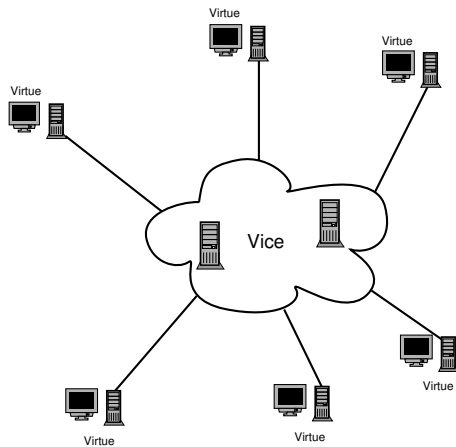
Das Coda Dateisystem (1/4)

- ▶ entwickelt seit 1987 an der Carnegie Mellon University
- ▶ experimenteller Nachfolger des Andrew File System Version 2 mit Fokus auf Verfügbarkeit
- ▶ Designprinzipien
 - ▶ Clients haben Rechenzeit übrig
 - ▶ Cachen, wo es möglich ist
 - ▶ Anpassung an die Nutzungsprofile der einzelnen Dateien (read-only, temporäre Dateien, ...)
 - ▶ Minimierung systemweit notwendiger Informationen und Änderungen
 - ▶ Möglichst kleine Menge vertrauenswürdiger Objekte
 - ▶ Zusammenfassung von Operationen, wo möglich

Das Coda Dateisystem (2/4)

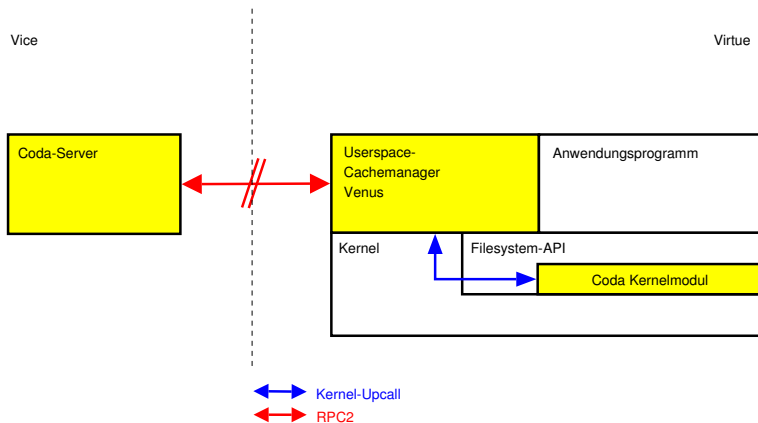
- ▶ Aufteilung des Systems in vertrauenswürdige Server (Vice) und nicht vertrauenswürdige Clients (Virtue)
- ▶ persistenter Cache auf Datei-Ebene auf dem Client zur Steigerung der Performanz und Verfügbarkeit
- ▶ *disconnected operation* – Arbeit auf Daten im Cache bei Verlust der Verbindung zum Server und transparente Reintegration von Änderungen nach dem erneuten Verbindungsaufbau
- ▶ Serverreplikation auf Volume-Ebene mit mehreren schreibfähigen Kopien (*Volume Storage Group*)
- ▶ Propagierung von Änderungen beim Schließen von Dateien (Vergleich UNIX-Semantik: as soon as possible, meist bei `write(...)`)

Das Coda Dateisystem (3/4)



Struktur einer Coda-Installation

Das Coda Dateisystem (4/4)



Struktur des Virtue

Einführung

Begriffsklärung

Problemstellungen bei verteilten Dateisystemen

Unterteilung verschiedener Dateisystemtypen

Vorstellung Coda Filesystem

Ansätze zur Steigerung der Skalierbarkeit

Replikation

Caching

Callbacks

Hints

Zentrale Nutzerverwaltung

Gruppen/Rechtevererbung

Negative Rechte

Fazit

Lastverteilung und Verfügbarkeitssteigerung durch Replikation

- ▶ mehrfache Datenhaltung auf verschiedenen Servern
- ▶ Synchronisation der Kopien untereinander
- ▶ Verteilung der Anfragen auf die verschiedenen Server

Vorteile:

- ▶ Einfache Erweiterbarkeit
- ▶ bei effizienter Verteilung der Anfrage nahezu lineare Skalierung mit der Anzahl der Serverknoten

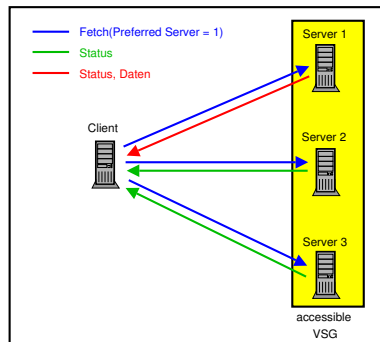
Nachteile:

- ▶ komplexe Synchronisation bei mehreren schreibfähigen Kopien
- ▶ erhöhtes Kommunikationsaufkommen durch Synchronisation der Server
- ▶ steigender Sicherungsaufwand für sensible Daten

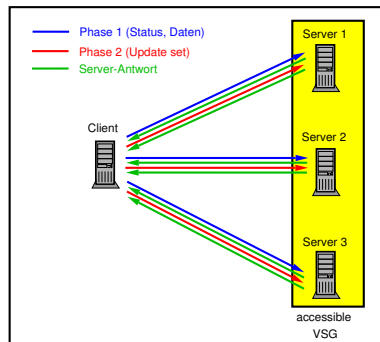
Replikation am Beispiel Coda (1/2)

- ▶ Replikation auf Volume-Ebene mit mehreren schreibfähigen Repliken (*Volume Storage Group*)
- ▶ aktuell verfügbare Repliken bilden die *accessible VSG*
- ▶ *read-one-data, read-all-status* Lesestrategie (durch *preferred server*), *write-all* Schreibstrategie
- ▶ Leseoperation mit *Cache Miss*
 1. parallele Anfrage an alle Server des AVSG mit *preferred server* als Parameter
 2. *preferred server* antwortet mit Status und Daten, andere Server nur mit Status
- ▶ Schreiboperation
 - Phase 1 Ausführen aller relevanten Operationen (Datei anlegen, schreiben, ...)
 - Phase 2 Übertragung des *update set* (Verwaltungsstruktur mit Versionsinformationen etc.)

Replikation am Beispiel Coda (2/2)



Leseoperation mit *Cache Miss*



Schreiboperation

Caching

- ▶ *Grundprinzip*: Vorhaltung zu bearbeitender Daten in einem kleinen, schnellen Zwischenspeicher, längerfristige Speicherung auf einem großen, langsameren Speichersystem
- ▶ allgemein verwendetes Prinzip, meist zur Performanzsteigerung (CPU-Cache, Browser-Cache, Festplatten-Cache), seltener zur Erhöhung der Verfügbarkeit (nichtflüchtiger Cache bei Coda Disconnected Operation)

Vorteile:

- ▶ hohe Geschwindigkeit
- ▶ transparent für laufende Applikationen

Nachteile:

- ▶ Algorithmen zur Konsistenzhaltung notwendig

Caching bei Coda

- ▶ Caching auf Datei-Ebene
- ▶ persistenter Cache
- ▶ Spezialfall *disconnected operation*: Cache zur Steigerung der Verfügbarkeit, indem vorhandene Dateien aus dem Cache bedient werden, Änderungen aufgezeichnet und später mit dem Vice integriert werden (transparent für den Nutzer)
- ▶ Änderungen werden beim Schließen einer Datei für andere Nutzer sichtbar (Gegensatz dazu: UNIX-Semantik – as soon as possible)
- ▶ Wahrung der Cache-Konsistenz durch *Callbacks*

Callbacks

- ▶ Möglichkeit zur Sicherstellung der Übereinstimmung zwischen den Daten in Cache und auf dem Server
- ▶ “Versprechen” des Servers über die Gültigkeit der Daten im Cache
- ▶ Modifikation der Daten auf dem Server bricht dieses Versprechen (*callback break*) → Information an den Client

Vorteile:

- ▶ drastische Senkung der Netzlast

Nachteile:

- ▶ erhöhter Ressourcenbedarf auf dem Server (Halten einer Statusinformation für jeden Client)

Callbacks bei Coda

- ▶ Übergabe eines *callback token* von Server zu Client beim Öffnen einer Datei
- ▶ statusbehafteter Server, der alle ausgegebenen Callbacks hält
- ▶ Schließen der Datei durch einen Client ($\hat{=}$ Schreiben von Änderungen) \Rightarrow Information an alle Clients, welche Callbacks auf diese Datei halten
- ▶ Client kann solange auf die Daten im Cache vertrauen, bis er einen *callback break* empfängt

Hints

Information, welche

- ▶ bei Korrektheit einen Vorgang beschleunigen
- ▶ bei Fehlerhaftigkeit *nicht* zu falschen Ergebnissen führen

⇒ Caching von Hints ist ohne Konsistenzsicherung möglich

Vorteile:

- ▶ bei korrektem Einsatz starke Senkung des Kommunikationsaufkommens und Steigerung der Geschwindigkeit

Nachteile:

- ▶ nicht alle Daten als Hints geeignet (müssen bei der Benutzung automatisch auf Korrektheit überprüft werden können)

Hints im Coda Filesystem

Caching von Einträgen aus der *volume location database*
(Zuordnung zwischen Pfadinformationen im *shared namespace* und Ortsinformationen im Vice)

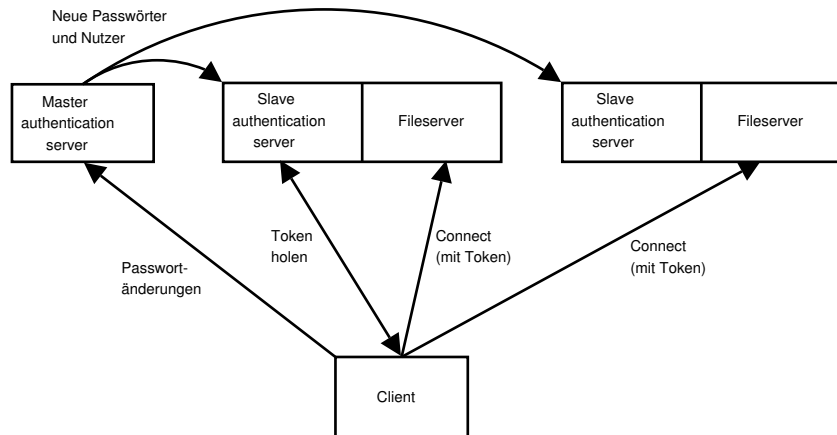
- ▶ ändern sich selten → überwiegend korrekt
- ▶ selbstvalidierend bei Verwendung (wenn ungültig, kann nicht auf die Datei zugegriffen werden und es erfolgt eine erneute Abfrage der Ortsinformation beim Server)

Zentrale Nutzerverwaltung

Problem: Verteilung von Nutzerinformationen erschwert Administration (“Auf welchem Server liegt der Nutzer?”)

- ▶ Zentralisierung der Nutzerverwaltung notwendig
- ▶ Möglichkeiten
 - ▶ zentraler Login-Server, der begrenzt gültige *Token* ausstellt (Bsp.: Kerberos)
 - ▶ Public Key Infrastruktur
 - ▶ Replikation von Nutzerdatenbanken

Nutzerverwaltung in einer Coda-Installation



Struktur der Coda-Nutzerverwaltung

Nutzergruppen und Rechtevererbung

- ▶ Konzept zur Zusammenfassung von Nutzern mit gleichen Rechten
- ▶ Mitgliedschaft von Gruppen in weiteren Gruppen möglich
- ▶ Summe der Rechte aller Gruppen, in denen ein Nutzer Mitglied ist, bildet Rechte eines Nutzers (*Rechtevererbung*)

Vorteile:

- ▶ einfache Möglichkeit zur Abbildung der hierarchischen Struktur einer Organisation

Nachteile:

- ▶ teilweise schwierige Abwägung zwischen Anzahl der Gruppen und Menge der Rechte einzelner Nutzer

Nutzergruppen und Rechtevererbung bei Coda

- ▶ Implementierung eines Gruppenkonzeptes
- ▶ Nutzer kann Mitglied in mehreren Gruppen sein
- ▶ Gruppen können ihrerseits Mitglieder in weiteren Gruppen sein
- ▶ Rechte eines Nutzers bilden sich aus den Rechten aller Gruppen, in denen er direkt oder indirekt Mitglied ist
- ▶ Rechtevergabe via *Access Control List (ACL)* für jede Subjekt-Objekt-Kombination einzeln möglich

Negative Rechte

- ▶ Möglichkeit zum Entzug bereits zugestander Rechte (bspw. durch Gruppenzugehörigkeit)
- ▶ werden auf bestimmte Objekte angewandt (in der ACL verankert)

Vorteile:

- ▶ einfacher Rechteentzug auf bestimmte Objekte

Nachteile:

- ▶ führen bei übermäßigem Einsatz zu unübersichtlicher Rechtesituation (kein Ersatz für Aufteilung der Nutzer in Gruppen)

Fazit

- ▶ Skalierbarkeit verteilter Dateisysteme muss sowohl *Last-* als auch *administrative Skalierbarkeit* betrachten
- ▶ Grundprinzipien zur Last-Skalierbarkeit
 - ▶ Verteilung nicht sicherheitsrelevanter Aufgaben an den Client
 - ▶ Caching, Replikation
 - ▶ Anpassungsfähigkeit an Nutzungsprofile
 - ▶ Zusammenfassung von Operationen
- ▶ Grundprinzipien zur administrativen Skalierbarkeit
 - ▶ feingranulare Möglichkeiten zur Zugriffssteuerung
 - ▶ Mittel zur Abbildung hierarchischer Strukturen
 - ▶ Zentralisierung von Nutzerinformationen
- ▶ Beispiele Andrew File System und Coda zeigen durch Installationen mit mehreren tausend Nutzern die Wirksamkeit der Maßnahmen

Vielen Dank für die
Aufmerksamkeit!

Unterlagen: <http://www.slash-me.net/work.html>

Kontakt: Markus Brückner (vortrag@slash-me.net)