Moderne Dateisysteme

Hochleistungs-Ein-/Ausgabe

Michael Kuhn

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

2018-04-20

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

1 Moderne Dateisysteme
- Orientierung
- Moderne Dateisysteme
- ZFS
- Pools
- Datensicherheit
- Datenreduktion
- Weitere Funktionen
- Leistungsaspekte
- Ausblick und Zusammenfassung

2 Quellen

E/A-Schichten

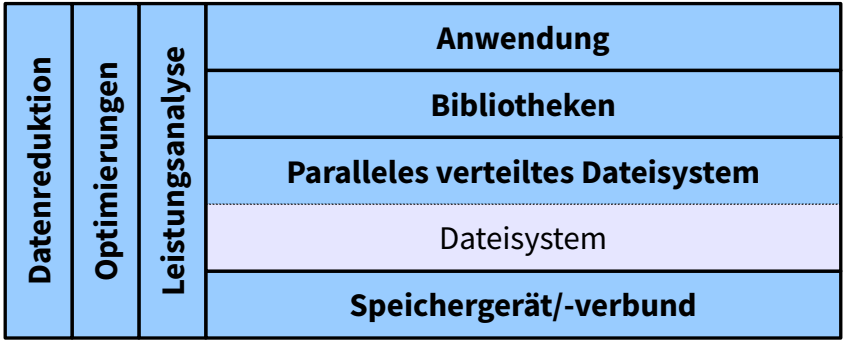


Abbildung: E/A-Schichten und orthogonale Themen

Rückblick

- Erinnerung: Dateisysteme zur Strukturierung
  - Verwaltung von Daten und Metadaten
  - Blockallokation, Zugriffsrechte, Zeitstempel etc.
- Dateisysteme nutzen ein/en Speichergerät/-verbund
  - Logical Volume Manager (LVM) und/oder mdadm
- Dateisysteme bieten nur grundlegende Operationen
  - Erstellen, Löschen, Lesen, Schreiben etc.

Überblick

- Anforderungen an Dateisysteme wachsen
  - Datenintegrität
  - Speicherverwaltung
  - Komfortfunktionen
- Fehlerrate bei SATA-Festplatten: 1 in 10^{14} bis 10^{15} Bits [1]
  - D. h. ein Bitfehler alle 12,5–125 TB
  - Zusätzlich Bitfehler im RAM, dem Controller, dem Kabel, dem Treiber etc.
- Fehlerrate kann problematisch sein
  - Datenmengen sind auch im täglichen Gebrauch erreichbar
  - Bitfehler kann auch im Superblock auftreten

Überblick…

- Dateisystem hat kein Wissen über Speicherverbund
  - Speicherverbund hat auch keine Informationen über Dateisysteminhalte
  - Ohne TRIM/DISCARD auch keine Informationen über Blockbelegung
- Wissen ist für optimale Leistung notwenig
  - Z. B. spezielle Optionen bei ext4
  - `-E stride=n,stripe_width=m`
- Hohe Wiederherstellungszeiten
  - Üblicherweise $\geq$ 10 h

- `stride` gibt die Anzahl der Dateisystemblöcke pro Speichergerät an
- `stripe_width` gibt die Anzahl der Dateisystemblöcke pro Stripe an
  - Stripe bezeichnet hier einen Streifen über die gesamte Breite
  - Üblicherweise $\texttt{stride} \cdot k$, wobei k die Anzahl der Speichergeräte ist, die Daten enthalten (ohne Parität)

Überblick…

- Zusätzliche Funktionalität
  - Schnappschüsse
  - Unterdateisysteme
  - Kompression
  - Verschlüsselung
  - Effiziente Backups
  - Deduplikation

- Unterdateisysteme können wie normale Verzeichnisse benutzt aber auch separat gemountet werden
  - In btrfs Subvolumes genannt

Überblick…

- Schnappschüsse
  - Z. B. zur effizienten Behandlung von Checkpoints
- Unterdateisysteme
  - Trennung unterschiedlicher Daten, unterschiedliche Konfigurationen
- Kompression und Deduplikation
  - Speicherdurchsatz und -kapazität können nicht mit Berechnungsgeschwindigkeit mithalten
- Verschlüsselung
  - Insbesondere im Unternehmensumfeld wichtig
- Effiziente Backups
  - Speichersysteme haben teilweise Größen im PB-Bereich

Allgemeines

- ZFS ist ein lokales Metadateisystem
  - Stand ehemals für „Zettabyte File System“
  - Integrierte Volumenverwaltung etc.
- Wurde initial von Sun Microsystems entwickelt
  - 2001: Entwicklungsbeginn
  - 2005: Veröffentlichung in OpenSolaris
  - 2006: Veröffentlichung in Solaris 10
  - 2008: ZFS-basierte Appliances
  - 2010: Oracle beendet Entwicklung als Open Source

Allgemeines…

- Aktuelle Entwicklungen
  - 2010: Abspaltung von illumos
  - 2013: OpenZFS-Initiative
  - 2013: Unterstützung als Lustre-Backend-Dateisystem
- Betriebssystemunterstützung
  - Solaris: Closed Source, inkompatibel zu OpenZFS
  - OS X: OpenZFS on OS X (O3X)
  - FreeBSD: Vollständige Unterstützung
  - Linux: ZFS on Linux (viele Distributionen)
- CDDL und GPL sind inkompatibel
  - Daher keine direkte Integration in Linux

Kompatibilität

- Versionierung der Dateisystemfunktionalität und des On-Disk-Formats
  - Aufsteigende Nummer vergeben durch Sun/Oracle
  - Durch Closed-Source-Weiterentwicklung eingeschränkte Kompatibilität zwischen ZFS und OpenZFS
- OpenZFS-Entwicklung mit Hilfe von „Feature Flags“
  - Version wurde auf 1000 bzw. 5000 festgelegt
  - Z. B. `async_destroy`, `lz4_compress`, `embedded_data` und `large_blocks`

- `async_destroy`: Dateisysteme werden im Hintergrund zerstört
- `lz4_compress`: lz4 steht als Kompressionsalgorithmus zur Verfügung
- `embedded_data`: Dateien, die (nach der Kompression) nicht größer als 112 Bytes sind, können im Blockzeiger gespeichert werden
- `large_blocks`: Blöcke können größer als 128 KiB werden
- Mehr Informationen in der `zpool-features`-Manpage

Features

- ZFS ist das weltweit erste 128-Bit-Dateisystem
  - 64 Bit sind ausreichend für 16 EiB
  - 128 Bit momentan physikalisch gar nicht ausnutzbar
  - *"Populating 128-bit file systems would exceed the quantum limits of earth-based storage. You couldn't fill a 128-bit storage pool without boiling the oceans."*
    – Jeff Bonwick, ehemaliger ZFS-Chefentwickler
- Datensicherheit unverzichtbar
  - Datenfehler werden automatisch erkannt und behoben
- Einfache Administration gepaart mit hoher Leistung
  - Komplette Administration mit zwei Werkzeugen möglich

Zahlen

- Maximale Objektanzahl pro Verzeichnis: 2^{48}
  - 2^{32} bei ext4 (pro Dateisystem)
- Maximale Größe einer Datei: 16 EiB (2^{64} Bytes)
  - 16 TiB bei ext4
- Maximale Größe eines Pools: 256 ZiB (2^{78} Bytes)
  - 64 ZiB bei ext4
- Maximale Geräteanzahl pro Pool: 2^{64}
- Maximale Poolanzahl: 2^{64}
- Maximale Dateisystemanzahl pro Pool: 2^{64}

Motivation

- Traditionelle Dateisysteme nutzen veraltete Konzepte
  - Kein Schutz gegen Datenfehler
    - ext4 kann nur Prüfsummen für Metadaten speichern
  - Hoher Administrationsaufwand
    - Geräte müssen zu Verbünden zusammengefasst werden
    - Geräte/Verbünde müssen partitioniert werden
    - Partitionen müssen formatiert werden
  - Unflexible Konzepte
    - Feste Block- und Dateisystemgrößen
    - Teilweise statische Datei-/Verzeichnisanzahlen

Designprinzipien

- Pools
  - Keine Festplatten, Partitionen etc. mehr
  - Pool stellt Speicherplatz für alle Dateisysteme bereit
- Datenintegrität
  - Wurde früher als zu teuer betrachtet
  - CPUs haben ausreichende Leistungsreserven
- Transaktionen
  - Daten sind immer konsistent
  - Zeitraubende Dateisystemüberprüfung entfällt

Pools

- Traditionell ein Dateisystem pro Partition
  - Volumenmanager um ein Dateisystem über mehrere Partitionen/Festplatten zu ermöglichen
- Aktuelle Dateisysteme sind sehr statisch
  - Größenänderungen sind eher problematisch
- Neues Poolkonzept
  - Nutze gesamte Kapazität und Bandbreite der Hardware
  - Halte die Dateisysteme dynamisch

Pools

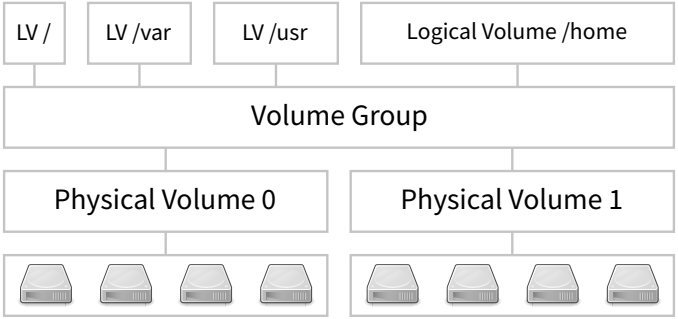


Abbildung: Traditionelle Architektur (RAID, LVM, Dateisystem)

Pools...

```
$ mdadm --create /dev/md0 --level=5 --raid-devices=4 /dev/sd[abcd]
$ mdadm --create /dev/md1 --level=5 --raid-devices=4 /dev/sd[efgh]
$ pvcreate /dev/md0
$ pvcreate /dev/md1
$ vgcreate tank /dev/md0 /dev/md1
$ lvcreate --size 15G --name root tank
$ lvcreate --size 25G --name var tank
$ lvcreate --size 30G --name usr tank
$ lvcreate --size 75G --name home tank
$ mkfs.ext4 /dev/mapper/tank-root
$ mkfs.ext4 /dev/mapper/tank-var
$ mkfs.ext4 /dev/mapper/tank-usr
$ mkfs.ext4 /dev/mapper/tank-home
```

Pools...

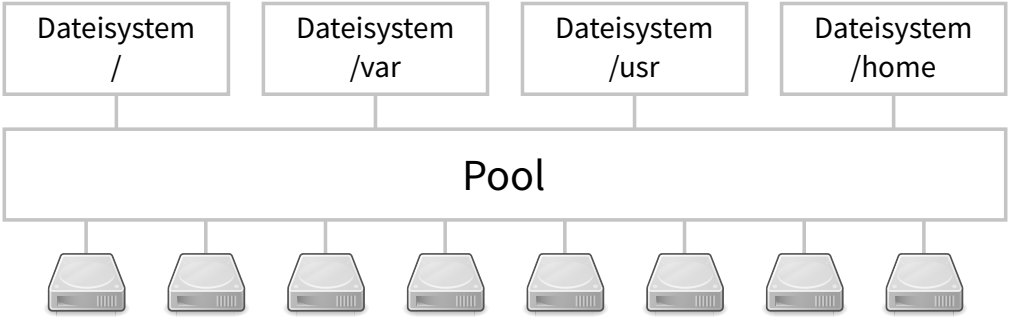


Abbildung: ZFS-Pool-Architektur

Pools...

```
$ zpool create tank raidz /dev/sd[abcd] raidz /dev/sd[efgh]
$ zfs create tank/root
$ zfs create tank/var
$ zfs create tank/usr
$ zfs create tank/home
```

Pools…

- Pools bestehen aus virtuellen Geräten (vdevs)
  - Daten werden dynamisch darüber verteilt
- Virtuelle Geräte können reale Geräte oder eine Zusammenfassung solcher sein
  - Mirror (RAID 1)
  - RAID-Z (RAID 5 ohne „Write Hole“)
  - RAID-Z2 (RAID 6 ohne „Write Hole“)
  - RAID-Z3 (ohne „Write Hole“)
- Erinnerung: „Write Hole“ ist die Zeit zwischen Schreiben der Daten und der Parität

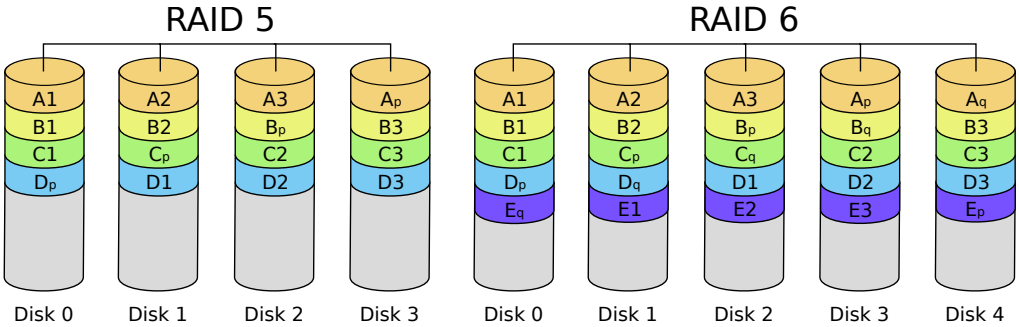
Dynamisches Striping

- Intelligente Verteilung der Daten auf alle virtuellen Geräte
- Mehrere Auswahlkriterien
  - Kapazität
  - Leistung (Latenz, Bandbreite, Auslastung)
  - Status (Mirror mit ausgefallener Festplatte)
- Neue virtuelle Geräte werden automatisch mitbenutzt
  - Existierende Daten werden nicht rebalanciert
  - Neues virtuelles Gerät wird bevorzugt

Dynamisches Striping…

1. Virtuelle-Geräte-Auswahl
   - Bevorzuge neue/leere virtuelle Geräte
   - Vermeide beeinträchtigte virtuelle Geräte
   - Ansonsten normales Round-Robin
     - Weitere Striping-Methoden evtl. in Zukunft
2. „Metaslab“-Auswahl
   - Bevorzuge die äußeren Regionen der Festplatten
   - Bevorzuge bereits benutzte Metaslabs
3. Block-Auswahl
   - Wähle den ersten Block mit genügend freiem Platz
     - Weitere Algorithmen evtl. in Zukunft

RAID 5 und RAID 6 [3]



(a) RAID 5 (b) RAID 6

RAID 5/6

- Daten und Parität müssen aktualisiert werden
  - Operationen auf mehreren Festplatten müssten atomar durchgeführt werden
  - Dadurch entsteht das „Write Hole“
- Schreiben von Streifen ist ineffizient
  - Read-Modify-Write, zwei Reads und zwei Writes
- Lösung: Hardware-RAID-Controllern mit großen Caches
  - Idee war ein Redundant Array of **Inexpensive** Disks

RAID-Z

- Das „Write Hole“ wird durch Copy on Write gepaart mit Transaktionen eliminiert
  - Damit sind atomare Aktualisierungen möglich
  - Normale Festplatten reichen aus
- Kein Schreiben von Teilstreifen
  - Jeder Block befindet sich in einem eigenen Streifen
  - Erhöht die Leistung, macht aber Rekonstruktion schwieriger
    - Dateisystemstruktur muss bekannt sein

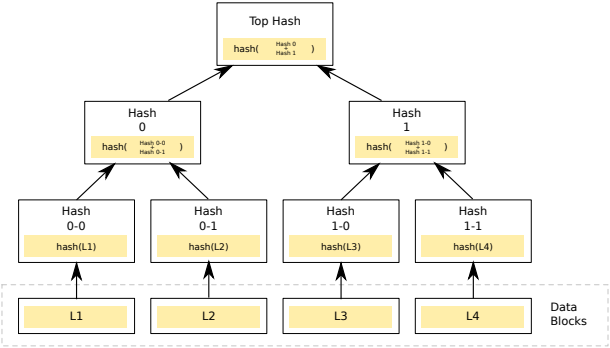
Transaktionen

- Operationen werden in Transaktionen durchgeführt
  - Auf Dateisystemebene alle Änderungen an Objekten
  - Auf Speicherebene alle Transaktionsgruppen (Daten und zugehörige Metadaten)
- ZFS befindet sich immer in einem konsistenten Zustand
  - Dadurch kein Journaling notwendig
  - Keine Dateisystemüberprüfung mehr notwendig

Dateisystemstruktur

- ZFS ist als Hash-Baum von Blöcken realisiert
  - Auch Merkle-Baum genannt
- Jeder Block enthält eine Prüfsumme
  - Es stehen mehrere Algorithmen zur Auswahl
- Bei jedem Lesen wird die Integrität des Blocks verifiziert
- Von Metadaten werden immer mehrere Kopien vorgehalten
  - Selbst ohne Redundanz durch mehrere Speichergeräte sollten Metadaten daher rekonstruierbar sein

Dateisystemstruktur… [2]

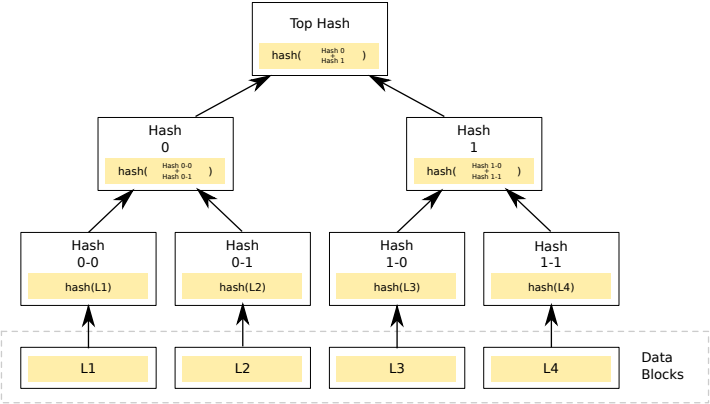


- Blätter enthalten Prüfsummen der Datenblöcke
- Andere Knoten enthalten Prüfsummen ihrer Kinder

Copy on Write

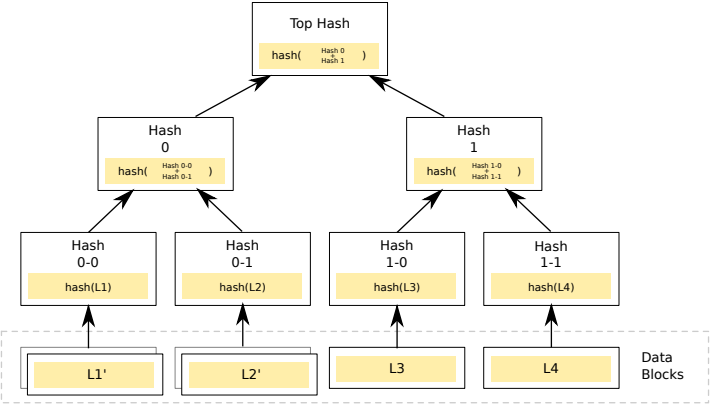
- Benutzte Blöcke werden nie überschrieben
  - Traditionell werden Blöcke direkt modifiziert
  - Stürzt währenddessen das System ab, sind die Daten möglicherweise inkonsistent
- Alle betroffenen Blöcke werden kopiert
  - Kopien werden modifiziert und geschrieben
  - Eigentlich Redirect on Write, üblicherweise aber als Copy on Write bezeichnet
- Alle Änderungen geschehen außerhalb des Dateisystems
  - Stürzt das System ab, sind gemachte Änderungen einfach nicht sichtbar und werden verworfen
- Zuletzt werden neue Blöcke atomar integriert

Copy on Write… [2]



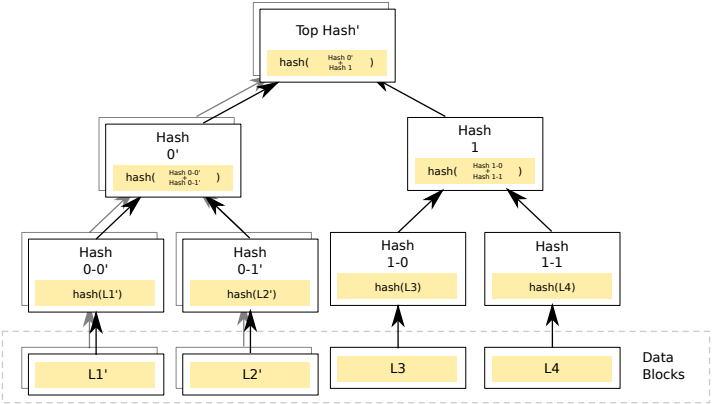
1 Ausgangszustand

Copy on Write… [2]



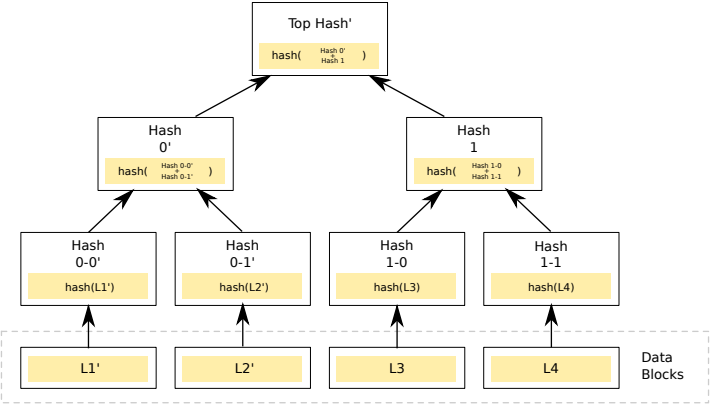
2 Neue Blöcke werden allokiert und mit Daten beschrieben

Copy on Write… [2]



3 Neue Zeigerblöcke werden allokiert und gesetzt

Copy on Write… [2]



4 Der „Überblock“ wird aktualisiert

Copy on Write…

- Überblock-Aktualisierung muss atomar geschehen
- ZFS hält ein Überblock-Array mit 128 Einträgen vor
  - Replikate des Arrays werden zur Sicherheit an mehreren Stellen im Pool gespeichert
  - Array wird im Round-Robin-Verfahren benutzt
- Überblocke enthalten Transaktionsnummer und Prüfsumme
  - Beim Mounten wird der Überblock mit der höchsten Transaktionsnummer benutzt
  - Integrität wird anhand der Prüfsumme überprüft

Schnappschüsse und Klone

- Schnappschüsse vereinfachen einige Anwendungsfälle
  - Vorhalten älterer Daten (z. B. tägliche Schnappschüsse)
  - Mehrere Checkpoints innerhalb einer Datei
    - Momentane Schnappschüsse sind dafür zu grobgranular
- Durch Copy on Write sehr einfach Schnappschüsse möglich
  - Alte Dateisystemwurzel als Schnappschuss speichern
  - Alte Zeiger und Blöcke nicht löschen
- Schnappschüsse können nur gelesen werden
  - Schnappschüsse sind im `.zfs`-Verzeichnis zu finden
  - Benutzer können so selbst auf Backups zugreifen

Schnappschüsse und Klone…

- Schnappschüsse können einfach zurückgerollt werden
  - Überblock ersetzen, ähnlich wie Transaktionen
  - Macht alle Änderungen seit dem Schnappschuss rückgängig
- Veränderbare Schnappschüsse nennen sich Klone
  - Unveränderte Blöcke werden geteilt
  - Änderungen an einem Klon durch neue Blöcke
- Klone sind ebenso einfach zu realisieren
  - Durch Copy on Write nur zusätzlicher Speicherverbrauch für neue und geänderte Daten

Backups

- Backups großer Speichersysteme sind problematisch
  - Traditionell Überprüfung des gesamten Namensraumes
  - Effizientere Abwicklung durch Schnappschüsse
- Vollständiges Backup
  - Als Grundlage dient ein beliebiger Schnappschuss
- Inkrementelles Backup
  - Grundlage sind Änderungen zwischen zwei Schnappschüssen
  - Aufwand hängt von den geänderten Daten ab
- Damit ist auch Replikation realisierbar
  - Erstelle minütlich Schnappschüsse und transferiere die inkrementellen Backups per SSH auf den anderen Server

RAID

- Fehlerszenario auf einem Mirror aus zwei Festplatten
  1. Programm liest Daten
  2. ZFS liest einen Block von der ersten Festplatte
  3. Es wird erkannt, dass der Block fehlerhaft ist
  4. ZFS liest die Blockkopie von der zweiten Festplatte
  5. Es wird erkannt, dass die Blockkopie korrekt ist
  6. Der fehlerhafte Block wird mit dem korrekten überschrieben
  7. Die Daten werden an die Anwendung weitergegeben
- Bei traditionellen Dateisystemen fallen die Schritte 3–6 weg
- ZFS kann erkennen, wenn beide Kopien fehlerhaft sind

RAID…

- Traditionelle RAID-Systeme können Fehler nur erkennen
  - Dafür müssten bei jedem Zugriff die Paritätsdaten gelesen und verglichen werden
  - Im Fehlerfall ist nicht klar ob Daten oder Parität korrekt sind
- Üblicherweise werden fehlerhafte Daten an die Anwendung weitergegeben
- Erkennung und Korrektur defekter Daten ist sehr wichtig
  - Teilweise sehr teuer zu berechnen
  - Aufgrund der Größe manchmal keine Backups

Scrubbing

- Scrubbing findet und korrigiert Fehler der Daten
- Für jeden Block wird Folgendes ausgeführt
  1. Block wird gelesen
  2. Block wird mit der gespeicherten Prüfsumme verglichen
  3. Falls der Block fehlerhaft ist, wird er wenn möglich repariert
- Scrubbing wird momentan nicht automatisch durchgeführt
  - Empfehlung: Wöchentlich bzw. monatlich

Resilvering

- Rekonstruktion der Daten in einem RAID-Verbund
- Traditionell musste alles rekonstruiert werden
  - Strenge Trennung zwischen Verbund und Dateisystem
  - Simples XOR über die noch vorhandenen Daten
  - Keine Möglichkeit die Korrektheit zu verifizieren
- Jetzt müssen nur noch tatsächlich vorhandene Blöcke rekonstruiert werden
  - Bei temporärem Ausfall nur zwischenzeitlich veränderte Daten
  - Mehr Datensicherheit durch Top-Down-Rekonstruktion
    - Verlust von Blöcken auf den oberen Ebenen fatal

- Die oberen Ebenen beziehen sich hierbei auf den Hash-Baum
  - Ein fehlerhafter Knoten macht den ganzen zugehörigen Teilbaum unzugänglich

Kompression

- Datenreduktion wird immer wichtiger
  - Speicherdurchsatz und -kapazität können nicht mit Berechnungsgeschwindigkeit mithalten
- ZFS unterstützt transparente Kompression
  - Kann auf Dateisystemebene gesetzt werden
  - Unterstützt mehrere Algorithmen
  - Momentan stehen zle, gzip, lzjb und lz4 zur Verfügung
- Kompression ist statisch
  - Gesetzter Algorithmus wird für alle Daten benutzt
  - Forschungsthema: Dynamische Kompression

Kompression…

- zle eliminiert Null-Sequenzen
  - Zero-Length Encoding
  - Üblicherweise geringe Kompressionsrate
  - Wird immer angewendet sobald Kompression aktiviert ist
- gzip komprimiert gut aber langsam
  - Unterstützt mehrere Komprimierungslevel (1–9)
  - Selbst schnelle Level recht langsam ($\approx$ 50 MB/s)
  - Dekompression schneller ($\approx$ 300 MB/s)

Kompression…

- lzjb wurde speziell für ZFS entwickelt
  - LZ: Lempel Ziv
  - JB: Jeff Bonwick (ehemaliger ZFS-Hauptentwickler)
  - Hohe Leistung
- lz4 schneller als lzjb
  - Hohe Kompressionsgeschwindigkeit ($\approx$ 600 MB/s)
  - Noch höhere Dekompressionsgeschwindigkeit ($\approx$ 3 GB/s)

Deduplikation

- Deduplikation ist ein Verfahren zur Datenreduktion
  - Daten werden in Blöcke aufgeteilt (statisch oder dynamisch)
  - Mehrfach vorkommende Blöcke werden nur einmal gespeichert
    - Weitere Vorkommen referenzieren Originalblock
  - Duplikate werden anhand der Prüfsumme erkannt
- Datensicherheit ist ein wichtiger Faktor in ZFS
  - Prüfsummen-Algorithmus wird auf SHA256 gestellt
  - Optional können Daten Byte für Byte verifiziert werden

Deduplikation…

- Deduplikation benötigt zusätzlichen Speicherplatz
  - Blöcke und Prüfsummen werden in Tabellen gespeichert
  - Tabellenzugriff bei jeder Schreiboperation
  - Tabellen sollten im Hauptspeicher gehalten werden
- Deduplikationsrate ist abhängig von der Blockgröße
  - Größere Blöcke verringern Deduplikationsrate
  - Kleinere Blöcke vergrößern Speicherbedarf
- 10+ GB pro TB bei einer Blockgröße von 8 KiB

Volumes

- ZFS unterstützt auch sogenannte Volumes
  - Exportiert als Blockgerät
  - Nützlich um andere Dateisystem auf Pools zu nutzen
- Alle Poolfunktionen können genutzt werden
  - Schnappschüsse, Komprimierung etc.

```
$ zfs create -V 4G tank/swap
$ zfs create -V 75G tank/home
$ mkswap /dev/zvol/tank/swap
$ mkfs.ext4 /dev/zvol/tank/home
```

Listing 1: ZFS-Volume

Diverses

- ZFS arbeitet mit eingeschaltetem Festplatten-Cache korrekt
  - Einzige Voraussetzung: Die Festplatten müssen die Flush-Kommandos befolgen
  - Bei einigen Dateisystemen kann Datenverlust auftreten
- ZFS nutzt dynamische Blockgrößen von bis zu 128 KiB
  - Kann auf 1 MiB erhöht werden
  - Die maximale Blockgröße kann manuell angepasst werden

Diverses…

- ZFS passt sich der Byte-Reihenfolge (Endianness) an
  - ZFS kümmert sich nur um Metadaten
  - Daten sind laut POSIX ein Stream von Bytes
- ZFS unterstützt mehrere unabhängige Prefetch-Ströme
  - Automatische Längen- und Schritt-Erkennung
  - Erkennt beliebige lineare Zugriffsmuster (vorwärts und rückwärts)

Regressionstests

- Lesen, Schreiben, Erstellen und Löschen von Dateien und Verzeichnissen
- Erstellen und Zerstören von Dateisystemen und Pools
- Aktivieren und Deaktivieren von Kompression
- Ändern des Prüfsummen-Algorithmus
- Hinzufügen und Entfernen von Geräten
- Ändern von Caching- und Scheduling-Strategien
- Zufällige Daten auf eine Hälfte eines Mirror schreiben
- Abstürze simulieren

“Probably more abuse in 20 seconds than you’d see in a lifetime.”
– Jeff Bonwick, ehemaliger ZFS-Chefentwickler

Einschränkungen

- Es ist nicht möglich einen Pool zu verkleinern
  - vdevs können nicht entfernt werden
- Es ist nicht möglich RAID-Z-vdevs zu erweitern
  - Geräte können durch größere ausgetauscht werden
  - Geräte können allerdings nicht hinzugefügt werden
- Nicht alle RAID-Level können abgebildet werden
  - Z. B. ist es nicht möglich ein RAID-51-Array zu erstellen
  - RAID 10, RAID 50 und RAID 60 sind allerdings möglich

Caching

- ZFS nutzt einen Pipeline-Scheduler
  - Jede Operation hat eine Priorität und eine Deadline
  - Höhere Priorität resultiert in kürzerer Deadline
  - Leseoperationen erhalten höhere Priorität als Schreiboperationen
- Operationen können zusammengefasst und umsortiert werden
  - Sonst bei jeder Änderung komplette Unterbaumkopie
  - Macht effizientes Copy on Write möglich

Caching…

- ZFS nutzt zwei Cache-Level: ARC und L2ARC
  - Der Adaptive Replacement Cache befindet sich im RAM
  - Der L2ARC befindet sich üblicherweise auf SSDs
- Alle Zugriffe werden durch den ARC beschleunigt
  - Dafür ist ausreichend viel RAM notwendig
  - Dort werden auch die Deduplikationstabellen gespeichert
- Befinden sich Daten nicht im ARC, wird auf den Pool zugegriffen

Caching…

- SSDs werden für den L2ARC und das ZIL genutzt
  - L2ARC wird nur für Lesezugriffe benutzt (inklusive der Deduplikationstabellen)
  - Das ZFS Intent Log ist journalartig und wird für synchrone Schreibvorgänge genutzt
- Schreib- und Lesecaches werden als Pool-Geräte verwaltet
  - Lesecaches als `cache`-vdevs
  - Schreibcaches als `log`-vdevs
  - Beide können hinzugefügt und wieder entfernt werden
- Gibt es kein separates Log, wird das ZIL im Pool angelegt

Overhead

- Komfortfunktionen erzeugen zusätzlichen Overhead
  - Prüfsummen müssen berechnet werden
  - Copy on Write erzwingt Read-Modify-Write
  - Kompression benötigt CPU-Leistung
  - Deduplikation benötigt CPU und RAM
  - Verschlüsselung benötigt CPU-Leistung

```
$ openssl speed sha256
type   16 bytes  64 bytes   256 bytes  1024 bytes 8192 bytes
sha256 79662.67k 183138.98k 337778.69k 423984.47k 461220.56k
```

Listing 2: SHA256-Geschwindigkeit (E3-1225 v3)

Overhead…

```
$ openssl speed aes-256-cbc
type    16 bytes    64 bytes    256 bytes   1024 bytes 8192 bytes
aes256 103533.94k 110310.34k 112363.45k 112561.83k 112787.46k
```

Listing 3: AES256CBC-Geschwindigkeit (E3-1225 v3)

```
$ openssl speed -evp aes-256-cbc
type    16 bytes    64 bytes    256 bytes   1024 bytes 8192 bytes
aes256 539161.24k 564210.73k 577473.93k 578992.13k 579510.27k
```

Listing 4: AES256CBC-Geschwindigkeit mit AES-NI (E3-1225 v3)

Overhead…

```
$ lz4 -1 -b fonts.tar
334929920 -> 243476163 (1.376), 526.8 MB/s, 2825.7 MB/s
$ lz4 -9 -b fonts.tar
334929920 -> 206036360 (1.626),  30.3 MB/s, 2360.9 MB/s
```

Listing 5: lz4-Geschwindigkeit (E3-1225 v3)

```
$ time gzip -1 -v fonts.tar
fonts.tar: 39.5%
 7,25s user 0,09s system 99% cpu  7,355 total (45 MB/s)
$ time gzip -9 -v fonts.tar
fonts.tar: 43.9%
19,67s user 0,09s system 99% cpu 19,797 total (17 MB/s)
```

Listing 6: gzip-Geschwindigkeit (E3-1225 v3)

Ausblick

- Moderne Funktionalität auch für parallele verteilte Dateisysteme sinnvoll
  - Prüfsummen insbesondere bei großen Datenmengen notwendig
  - Schnappschüsse/Versionierung wünschenswert
  - Lustre unterstützt sowohl ldiskfs als auch ZFS
- Data Management Unit in ZFS bietet effizienten Object Store
  - Kann durch andere Projekte genutzt werden
  - Erlaubt Trennung von Speicherallokation und eigentlicher Dateisystemfunktionalität

Zusammenfassung

- Dateisysteme organisieren Daten und Metadaten
- Moderne Dateisysteme integrieren zusätzliche Funktionen
  - Datensicherheit, Schnappschüsse, Datenreduktion etc.
- Copy on Write hilft Konsistenz zu bewahren
  - Daten werden nie überschrieben
- Integrierte Speicherverwaltung hat Vorteile
  - Erhöhte Leistung und Datensicherheit
- Kompression und Deduplikation können Speicherplatz sparen
  - Zusätzlicher CPU- bzw. RAM-Overhead

1 Moderne Dateisysteme
- Orientierung
- Moderne Dateisysteme
- ZFS
- Pools
- Datensicherheit
- Datenreduktion
- Weitere Funktionen
- Leistungsaspekte
- Ausblick und Zusammenfassung

2 Quellen

Quellen I

[1] Seagate. Desktop HDD. http://www.seagate.com/www-content/datasheets/pdfs/desktop-hdd-8tbDS1770-9-1603DE-de_DE.pdf.

[2] Wikipedia. Hash-Baum. http://de.wikipedia.org/wiki/Hash-Baum.

[3] Wikipedia. Standard RAID levels. http://en.wikipedia.org/wiki/Standard_RAID_levels.