

Moderne Dateisysteme

Hochleistungs-Ein-/Ausgabe

Michael Kuhn

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

2015-05-04

1 Moderne Dateisysteme

- Orientierung
- Moderne Dateisysteme
- ZFS
- Leistungsbewertung
- Ausblick und Zusammenfassung

2 Quellen

Rückblick

- Erinnerung: Dateisysteme zur Strukturierung
 - Verwaltung von Daten und Metadaten
 - Blockallokation, Zugriffsrechte, Zeitstempel etc.
- Dateisysteme nutzen ein Speichergerät/-verbund
 - Logical Volume Manager (LVM) und/oder mdadm

Überblick...

- Dateisystem hat kein Wissen über Speicherverbund
 - Und umgekehrt
- Wissen ist für optimale Leistung notwendig
 - Z.B. spezielle Optionen bei ext4
 - -E stride=n,stripe_width=m
- Hohe Wiederherstellungszeiten
 - Speicherverbund kennt Dateisystembelegung nicht
 - Üblicherweise ≥ 8 h

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

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, ZFS-Chefentwickler
- Datensicherheit unverzichtbar
 - Datenfehler werden automatisch erkannt und behoben
- Einfache Administration gepaart mit hoher Performance
 - Komplette Administration mit zwei Werkzeugen möglich



Zahlen

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



Pools

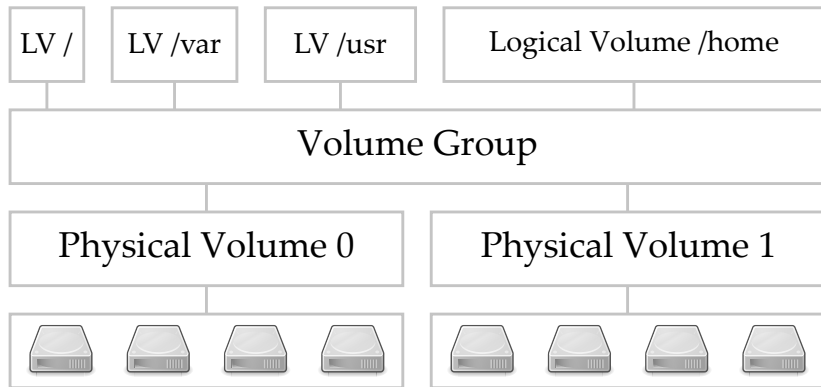


Abbildung: Traditionelle Architektur (RAID, LVM, Dateisystem)



Pools...

```
1 $ mdadm --create /dev/md0 --level=5 --raid-devices=4
   ↪ /dev/sd[abcd]
2 $ mdadm --create /dev/md1 --level=5 --raid-devices=4
   ↪ /dev/sd[efgh]
3 $ pvcreate /dev/md0
4 $ pvcreate /dev/md1
5 $ vgcreate tank /dev/md0 /dev/md1
6 $ lvcreate --size 15G --name root tank
7 $ lvcreate --size 25G --name var tank
8 $ lvcreate --size 30G --name usr tank
9 $ lvcreate --size 75G --name home tank
10 $ mkfs.ext4 /dev/mapper/tank-root
11 $ mkfs.ext4 /dev/mapper/tank-var
12 $ mkfs.ext4 /dev/mapper/tank-usr
13 $ mkfs.ext4 /dev/mapper/tank-home
```


Pools...

```
1 $ zpool create tank raidz /dev/sd[abcd] raidz /dev/sd[efgh]
2 $ zfs create tank/root
3 $ zfs create tank/var
4 $ zfs create tank/usr
5 $ 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 5/6

- Daten und Parität müssen aktualisiert werden
 - Operationen auf mehreren Festplatten müssten atomar durchgeführt werden
 - Dadurch entsteht das „Write Hole“
- Schreiben von Teilstreifen ist ineffizient
 - Read-Modify-Write
 - Warten auf Read
- 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 Dateisebene 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



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
- Alle Änderungen geschehen außerhalb des Dateisystems
 - Stürzt das System ab, sind gemachte Änderungen einfach nicht sichtbar
- Zuletzt werden neue Blöcke atomar integriert

Copy on Write...

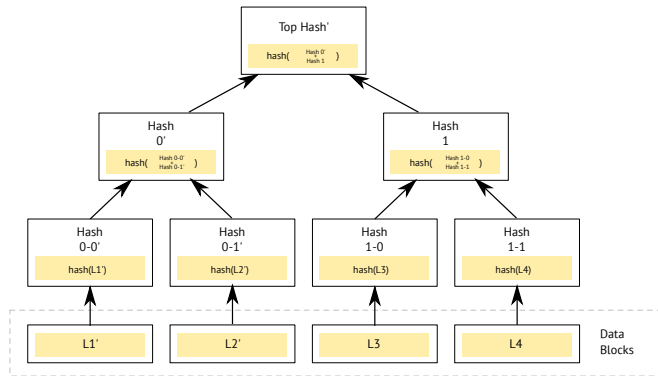


Abbildung: Copy on Write [1]

4 Der „Überblock“ wird aktualisiert

Copy on Write...

- Überblock-Aktualisierung
 - ZFS hält ein Überblock-Array mit 128 Einträgen vor
 - Replikate des Arrays werden an mehreren Stellen im Pool gespeichert
 - Array wird im Round-Robin-Verfahren benutzt
 - Überblock enthält u.a. eine Transaktionsnummer und Prüfsumme
 - Beim Mounten wird der Überblock mit der höchsten Transaktionsnummer benutzt
 - Integrität wird anhand der Prüfsumme überprüft

Schnappschüsse und Klone

- 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
 - 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

Backups

- Backups basieren auf Schnappschüssen
- Vollständiges Backup
 - Als Grundlage dient ein beliebiger Schnappschuss
- Inkrementelles Backup
 - Als Grundlage dienen die Änderungen zwischen zwei Schnappschüssen
 - Aufwand hängt von den geänderten Daten ab
- Damit ist Replikation realisierbar
 - Erstelle minütlich Schnappschüsse und transferiere die inkrementellen Backups per SSH auf den anderen Server

Datensicherheit

- **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

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
- Nur noch tatsächlich vorhandene Blöcke rekonstruieren
 - Bei temporärem Ausfall müssen nur veränderte Teile rekonstruiert werden
 - Mehr Datensicherheit durch Top-Down-Rekonstruktion
 - Verlust von Blöcken auf den oberen Ebenen fatal

Kompression

- 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

Kompression...

- zle eliminiert Null-Sequenzen
 - Zero-Length Encoding
 - Üblicherweise geringe Kompressionsrate
- gzip komprimiert gut aber langsam
 - Unterstützt mehrere Komprimierungslevel (1–9)
 - Selbst schnelle Level recht langsam (~ 50 MB/s)
 - Dekompression schneller (~ 300 MB/s)

Kompression...

- lzjb wurde speziell für ZFS entwickelt
 - LZ: Lempel Ziv
 - JB: Jeff Bonwick (ZFS-Hauptentwickler)
 - Hohe Leistung
- lz4 relativ neu und schneller lzjb
 - Hohe Kompressionsgeschwindigkeit (~ 400 MB/s)
 - Noch höhere Dekompressionsgeschwindigkeit ($\sim 1,8$ GB/s)

Deduplikation

- Deduplikation ist ein Verfahren zur Datenreduktion
 - Mehrfach vorkommende Blöcke werden nur einmal gespeichert und referenziert
 - Duplikate werden anhand der Prüfsumme erkannt
- Datensicherheit ist ein wichtiger Faktor
 - 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.

```
1 $ zfs create -V 4G tank/swap
2 $ zfs create -V 75G tank/home
3 $ mkswap /dev/zvol/tank/swap
4 $ mkfs.ext4 /dev/zvol/tank/home
```

Listing 1: ZFS-Volume

Leistungsaspekte

- 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
- Der Scheduler kann mehrere Operationen zusammenfassen und umsortieren
 - Macht effizientes Copy on Write möglich
 - Sonst müsste bei jeder Änderung ein kompletter Unterbaum kopiert werden

Leistungsaspekte...

- 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 die Daten nicht im ARC, wird auf den Pool zugegriffen

Leistungsaspekte...

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

Diverses

- ZFS arbeitet auch 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
 - Soll auf 1 MiB erhöht werden
 - Die maximale Blockgröße kann manuell angepasst werden

Einschränkungen

- Es ist nicht möglich einen Pool zu verkleinern
 - Geräte können nicht entfernt werden
- Es ist nicht möglich RAID-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

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

```
1 $ openssl speed sha256
2 type      16 bytes      64 bytes      256 bytes     1024 bytes    8192 bytes
3 sha256    54163.05k     122745.00k    217991.85k    263372.12k    277577.73k
```

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

Overhead...

```
1 $ openssl speed aes-256-cbc
2 type      16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes
3 aes256    104206.79k    110834.13k    112380.94k    113369.77k    113360.90k
```

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

```
1 $ openssl speed -evp aes-256-cbc
2 type      16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes
3 aes256    529875.27k    558823.38k    565729.88k    568951.47k    567079.63k
```

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

Zusammenfassung

- Dateisysteme organisieren Daten und Metadaten
- Moderne Dateisysteme integrieren zusätzliche Funktionen
 - U.a. verbesserte Datenintegrität, Schnappschüsse, Datenreduktion etc.
- Copy on Write hilft Konsistenz zu bewahren
- Integrierte Speicherverwaltung hat Vorteile bezüglich Leistung und Datensicherheit
- Kompression und Deduplikation können dabei helfen Speicherplatz zu sparen

1 Moderne Dateisysteme

- Orientierung
- Moderne Dateisysteme
- ZFS
- Leistungsbewertung
- Ausblick und Zusammenfassung

2 Quellen



Quellen I

[1] Wikipedia. Hash-Baum.

[http://de.wikipedia.org/wiki/Hash-Baum.](http://de.wikipedia.org/wiki/Hash-Baum)