
ORACLE®

ORACLE®

Btrfs und ZFS
Eine Gegenüberstellung von Dateisystemen der neuen Generation

Lenz Grimmer, Senior Product Manager, Oracle Linux

Ulrich Graef († 2013-06-04)

Überblick / Historie

Gemeinsamkeiten – Btrfs und ZFS

- Dateisystem plus Volume Manager
  - Mehrere Platten pro Dateisystem möglich
- Copy on Write
- Logging
- Snapshots
- RAID
- Kompression
- Checksummen
- **Aber:** unterschiedliche Implementierungen!

Überblick / Historie – Btrfs

- Ein modernes Dateisystem für Linux
- Initiiert und koordiniert von Chris Mason (FusionIO)
- Gemeinsam entwickelt durch Beiträge von
  - Fujitsu, FusionIO, Huawei, Intel, Oracle, Red Hat, Strato, SUSE u.a.
- Open Source (GPL)
- In mainline Linux seit 2.6.29 (Jan. 2009)

Überblick / Historie – ZFS

- Ein Dateisystem einer neuen Generation
- Entwicklung für Solaris durch Sun, nun Oracle
  - Lead: Jeff Bonwick
  - Clean Room Entwicklung; ohne bisherige FS Entwickler
  - Wenige Beiträge aus der Community via OpenSolaris
- Lizenz CDDL, letzte Publikation schon einige Zeit her
- Portierungen zu FreeBSD, Linux, Mac OS X

Architektur und Features

Features ZFS

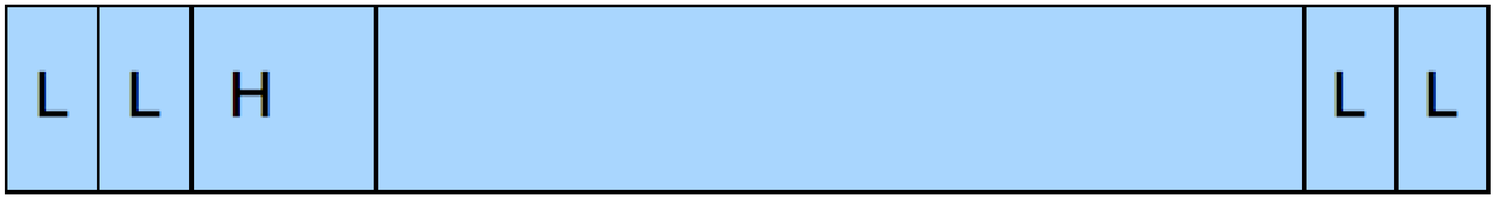
- Veränderungen auf Disk nur transaktionell
  - Daten und Metadaten durch gemeinsame CoW Transaktion
    - Mit 128-fachem Shadow Copy auf dem uberblock
  - Synchrones Schreiben durch Intent-Log
  - RAID (1, 2, 3 disk redundant) und Mirror (n-fach) möglich
- Alle Blöcke mit langen Prüfsummen gesichert
  - Validierender Baum (*Strukturelle Integrität*)
    - Prüfsumme des Blocks steht bei Pointer auf Block
- Lesen (Cache) und Schreiben (Log) separat mit SSD tunbar
- Kompression (lzjb und zlib)

Features ZFS

- De-Duplikation optional einschaltbar
- Encryption (mit Einbindung in das PAM System)
- Multiple ZFS in einem ZPool (wie disk pool)
  - Freigegebener Platz sofort in anderem ZFS nutzbar
- Eigenschaften durch Attribute steuerbar
  - Kompression, Block-Größe, Cache-Nutzung, Quota, Reservierung, Mount-Punkt, Export (NFS, CIFS, iSCSI, ...)
  - Eigene Attribute (Textfelder) möglich (Besitzer, Kostenstelle, ...)
- Hierarchische Vererbung der Attribute (Vereinfachung)
- Delegation an andere User und/oder an Zonen

Architektur und Features - ZFS On Disk Layout

- Physikalisches On-Disk Layout
  - Keine Sonderstrukturen mehr (wie Zylindergruppen, superblocks)
  - Physikalische Struktur auf jeder Platte: Label (4x), *Hole*, Blocks



    - Label enhält uberblock und Konfiguration des Pools
      - Redundant mit Checksummen
    - Blocks bildet den Platz des Zpools
      - ggf: Mirror oder Raidz

Architektur und Features - ZFS Logische Struktur

- Logische Struktur
  - Am uberblock hängt ein Baum von ZFS Datasets (Filesystemen)
    - Oder auch ein Volume (erscheint als Device unter /dev)
  - Jeder ZFS Dataset hat
    - Attribute
    - Snapshots (read-only, für Compliance...)
    - Clones (read-write)
    - Dnode-Liste (wie inode, nur 1 Eintrag für Volumes)
    - Block Allokation ist ein selbst-validierender Baum (Grad 1024)
      - 128KByte Block-Größe / 128 Byte Referenz

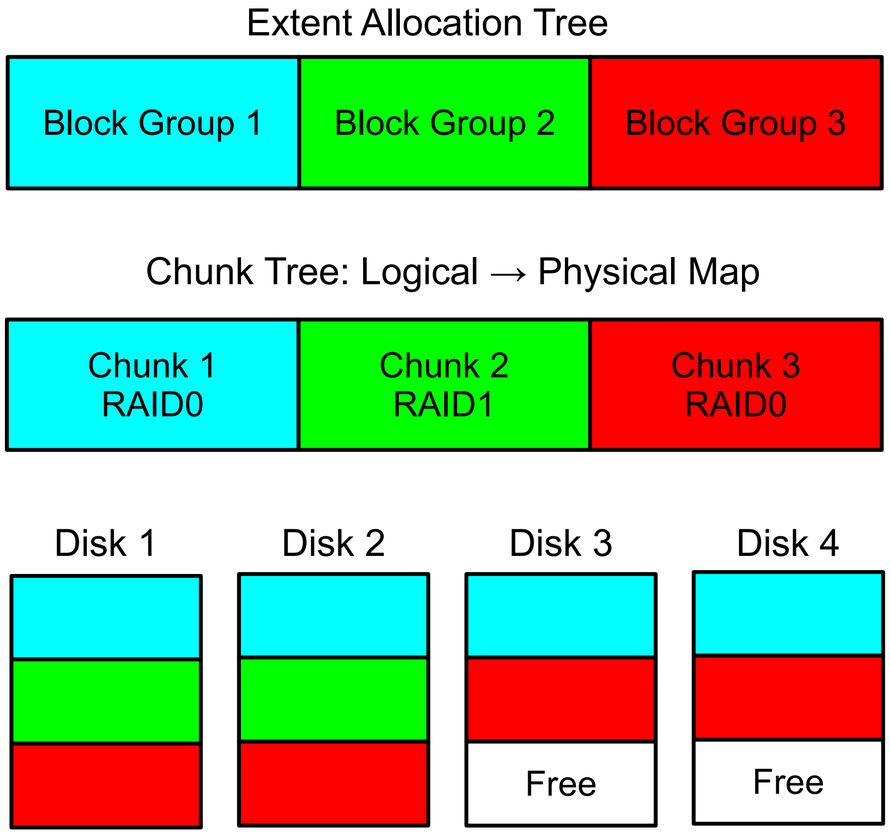
Features – Btrfs

- Schreibt Daten und Metadaten via copy-on-write (COW)
- Checksummen (CRC32C) für alle Daten und Metadaten
- Effiziente Snapshots (beschreibbar oder nur-lesend)
- RAID-Unterstützung (RAID0/1/10)
- Online Größenänderung und Defragmentierung
- Transparente Kompression (zlib/LZO)
- Effiziente Speicherung kleiner Dateien
- SSD Optimierungen und Discard/TRIM support

Architektur – Btrfs

- B-Baum als fundamentale Datenstruktur
- B-Bäume speichern beliebige Schlüssel/Wert-Paare
- Metadatenstrukturen verwenden bestimmte Schlüssel um verwandte Elemente nah beieinander zu speichern
- Speicherplatz wird in Blockgruppen verwaltet
- Blockgruppen werden in Chunks unterteilt
  - 1GB für Daten
  - 256 MB für Metadaten

Architektur – Btrfs – Speicherzuweisung



- Chunk Tree verwaltet Verteilung auf Disks
- Daten und Metadaten können verschiedene RAID Level haben

Fehlertoleranz / Datenintegrität

Fehlertoleranz / Datenintegrität – Btrfs

- Checksummen für Daten und Metadaten
  - CRC-32C (andere möglich)
- Btrfs Scrub scannt Daten- und Metadatenblocks
  - Automatische Korrektur (wenn intakte Kopie existiert)
- RAID 0/1/10
  - Verschiedene RAID-Level für Daten/Metadaten
  - Chunk-basiert, schnelle Wiederherstellung
  - (RAID 5/6 in Entwicklung)
- `mount -o recover`, `btrfschk` und `btrfs-restore`

Fehlertoleranz / Datenintegrität ZFS

- Prüfsummen liegen bei den Pointern
  - 128Byte Pointer mit 128 oder 256 Byte Prüfsumme
    - Komplexe SHA-1 oder MD-5, einfache: Fletcher-2 und Fletcher-4
  - Rechenzeit bei heutigen CPUs nicht mehr relevant (wenige µSek)
  - Vorteil: Defekte Blöcke werden nie benutzt!
  - Nicht mögklich mit dem klassischen System (Spiegeln im VolMgr)
- scrub statt fsck
  - Filesystem fsck geht in der Regel nur offline
  - scrub geht jederzeit, auch bei gemountetem Filesystem
  - fsck verlässt sich auf die Daten / scrub überprüft die Prüfsummen

Skalierbarkeit / Limitierungen

Skalierbarkeit / Limitierungen – ZFS

- Max. Dateigröße: 8 EiB
- Max. Dataset Größe: 2^{128} Bytes
- Deduplizierung nur auf Blöcken (recordsize)
- Konfigurierbare recordsize von 512 Byte bis max 1MByte
- RAIDZ optimiert auf Durchsatz (Blöcke werden aufgeteilt)

Skalierbarkeit / Limitierungen – Btrfs

- Max. Dateigröße: 8 EiB
- Max. Anzahl Dateien: 2^64
- Max. Volumengröße: 16 EiB
- Max. Dateinamen-Länge: 255 Byte

- Noch keine (interne) Deduplizierung oder Verschlüsselung
- Behandlung von ENOSPC verbesserungswürdig

Administration

Administration – Btrfs

- Kommandozeile, GUIs in Arbeit (z.B. btrfs-gui, YaST)
- `mkfs.btrfs, btrfs-convert`
- `btrfs <subcommand>`
  - `btrfs device`
  - `btrfs subvolume`
  - `btrfs filesystem`
- Mount-Optionen
- `btrfsck, btrfs-restore`

Spezialitäten – Btrfs

- Migration bestehender ext3/4-Dateisysteme
- Seed Devices
- (Inkrementelle) Backups mit `btrfs send/receive`
- Änderung der RAID-Level für Daten/Metadaten zur Laufzeit
- Snapshots von Dateien (`cp --reflink`)
- Schnelle Backups mit `btfs subvolume find-new`
- Alternative Kompressionsalgorithmen

Administration – ZFS

- `zpool create` Create a pool and a filesystem
- `zpool add` Enlarge a pool
- `zpool destroy` Remove a pool
- `zpool export`
- `zpool import` Access to a pool

- `zfs create` Create a ZFS
- `zfs create -V` Create volume
- `zfs set` Change Attributes
- `zfs snapshot` Take snapshot
- `zfs clone` Writable copy
- `zfs rollback` Reset to snapshot
- `zfs send` Serialize a snapshot
- `zfs receive` Recreate from serial form

ZFS Spezialitäten

- Extended Attributes
  - In Linux: kleine Datenstücke (Btrfs: 3.9k), für ACLs benutzt
  - In Solaris / BSD: Jedes Attribut so gross wie eine Datei (8 EB)
- `zfs send` / `zfs receive`
  - Serialisierung eines Snapshots (auch inkrementell)
  - Für Migration, Replikation (asynchron)
- ZFS virus scan interface
  - API zum Zugriff für Virenscanner

ZFS und Btrfs Kostbarkeiten

- Snapshot Daten sind für User sichtbar
  - Restore für User wird einfacher, gerade im Bereich Development
- Transaktionelles Schreiben
  - Unverzichtbar für grosse Filesysteme (> n TByte)
  - Ohne Transaktionen (UFS, ext*): fsck aufwendig (Tage, Wochen)
  - Restart nach Crash unzumutbar: nicht für Produktion

Links, Ressourcen

Links, Ressourcen

- Btrfs Wiki: https://btrfs.wiki.kernel.org/
- Mailing List: http://vger.kernel.org/vger-lists.html#linux-btrfs
- #btrfs channel IRC (freenode.net)

- ZFS Dokumentation

  http://docs.oracle.com/cd/E26502_01/html/E29007/index.html
- ZFS at OpenSolaris.org (demnächst move)

  http://hub.opensolaris.org/bin/view/Community+Group+zfs/

Hardware and Software

ORACLE

Engineered to Work Together


ORACLE®