Die Anatomie von ZFS

Workshop
Tübix 2017-06-24

Einführung

- ZFS bietet
  - Speicherverwaltung in redundanten Storage-Pools
  - hierarchische Objektstruktur mit Transaktionssemantik
  - inhärente Integritätssicherung
  - flexible Dateisystemverwaltung
  - skalierbare Snapshot- und Replikationsmechanismen
  - adaptives Cache-Management

Einführung

- ZFS bietet
  - Speicherverwaltung in redundanten Storage-Pools
  - hierarchische Objektstruktur mit Transaktionssemantik
  - inhärente Integritätssicherung
  - flexible Dateisystemverwaltung
  - skalierbare Snapshot- und Replikationsmechanismen
  - adaptives Cache-Management

ZFS zum Anfassen

- ZFS-Funktionsweise praktisch nachvollziehen
- Zwei Kommandos für ZFS:
  - zpool
  - zfs
- Verwenden hauptsächlich das dritte: zdb
- Werkzeug für Eingeweihte
- ungewöhnliche Syntax:
  - höhere Detailtiefe durch wiederholte Optionen
  - völlig normal: zdb -bbbbb

Baumstruktur, Block Pointer und Objekte

- Experimente mit: zdb -b (Baumstruktur), zdb -u (Uberblock)
- einfachen Pool anlegen (1 Disk), ein Dateisystem anlegen
- Baumstruktur analyisieren:
  - Uberblock
  - Objects, Object Sets
  - Block Pointer
  - Prüfsummen
  - DVAs

Baumstruktur: Lessons Learned

- ZFS verwaltet sämtliche Inhalte eines Pools in Objekten einer hierarchischen Baumstruktur
- Gruppierung von Objekten in Object Sets
- Objekt 0 eines Object Sets verweist von Objektnummer zu Speicherplatz
- Block-Pointer bildet Struktur ab
  - innerhalb eines Objekts (indirekte Blöcke)
  - zwischen Objekten (dnode-Arrays)
  - enthält Prüfsumme des Blocks, auf den er verweist
- Uberblock als Wurzel der Baumstruktur

Transaktionen und Copy on Write

- Experimente mit:
  - Datei anlegen (dd), dann zdb -b, zdb -u
  - Inhalte überschreiben
  - zdb -b, zdb -u, diff zwischen beiden Ausgaben
- Funktionsweise Copy on Write (aka Redirect on Write)
  - neue Inhalten erhalten neuen Speicherplatz
  - Änderung auf Leaf-Ebene erzwingt rekursiv Änderung auf übergeordneter Ebene im Baum
  - Schreiben der neuen Wurzel (Uberblock) schließt Transaktion (Transaktionsgruppe) ab
- frühere Uberblocks bilden konsistenten früheren Zustand ab
- Rücksetzen begrenzt möglich (zdb -T/-X)

Copy on Write: Lessons Learned

- ZFS platziert Schreibzugriffe auf freie Poolbereiche
- alte Inhalte werden nie (unmittelbar) überschrieben, aber ggf. freigegeben
- jede Änderung erzwingt Änderungen der Baumstruktur bis zur Wurzel (Uberblock)
- ZFS gruppiert mehrere Transaktionen zu Transaktionsgruppen
- kein Einfluss auf Speicherzuweisung innerhalb Pool
  -> ähnliche Hardware sinnvoll
  -> ähnliche Redundanzverfahren sinnvoll
- Online-Konsistenzprüfung über Scrubbing

Datasets, Volumes, Dateisysteme

- Experimente mit: zdb -d
- Objekte mit unterschiedlichen Typen
  - zur Verwaltung der Poolstruktur
  - zur Verwaltung von Nutzdaten
    - als Dateisystem (ZFS)
    - als Blockgerät (ZVOL)
  - Objekt-Typ bestimmt Interpretation der Objekt-Inhalte
- Dataset: Object-Set-Typ bestimmt Interpretation der Object-Set-Inhalte

Datasets: Lessons Learned

- ZFS-Dateisystem implementiert über Objekte unterschiedlichen Typs:
  - 1 Datei = 1 Objekt
  - 1 Verzeichnis = 1 Objekt
- Verzeichnis: Liste von Name-Objektnummer-Paaren
- Unterscheide:
  - Poolstruktur: Verweise über Block-Pointer
  - Dateisystem-Struktur: Verweise über Objektnummer
  - Pool- und Dateisystemstruktur voneinander unabhängig
- Vorgaben für Replikation von Objekten nach Bedeutung:
  - Uberblock: 4x
  - Poolstruktur: 3x
  - Datasetstruktur: 2x
  - Nutzdaten: 1x

Birth txg, Snapshots, asynchrone Replikation

- Experimente mit: zdb -d, zstreamdump, zfs diff
- Zeitliche Ordnung aller Änderungen über Transaktionsgruppennummer (txg)
- Snapshot:
  - Behalte Verweis auf Dataset-Wurzel zum Zeitpunkt des Snapshots
  - Lösche keine Blöcke, die vor dem Zeitpunkt des jüngsten Snapshots entstanden sind (-> Dead List)
- (inkrementelle) Replikation
  - Bilde konsistentes Abbild zu einem festen Zeitpunkt (Snapshot) als neues Ziel
  - Finde txg-Nummer des Ausgangszeitpunkts (Snapshot oder Bookmark)
  - Durchlaufe Baumstruktur (über Block Pointer)
  - Vergleiche birth txg mit txg-Nummer der letzten Replikation sowie dem Ziel-Snapshot
  - Sende alle Blöcke mit birth txg zwischen Ausgangs- und Zielzeitpunkt

Birthtimes: Lessons Learned

- ZFS kennt Lebensdauer eines Blocks über
  - Birth txg
  - Dead List
- Zusatzaufwand für Snapshots nur
  - beim (expliziten oder impliziten über CoW-Aktualisierungen) Löschen von Blöcken im Dateisystem/Volume (gering)
  - beim Löschen von Snapshots (Verarbeitung Dead List, ggf. hoch)
- effiziente Replikation über Poolstruktur (zfs send)
- ineffizient nur bei (optionaler) Übersetzung auf Dateisystemstruktur (zfs diff)
- Bookmarks als leichtgewichtige Alternative zu Snapshots für Ausgangspunkt von Replikation

Synchrone Operationen, NFS, ZIL/slog

- Experimente mit:
  - tar-Archiv in lokales ZFS-Dateisystem entpacken
  - tar-Archiv in loopback-NFS-Mount desselben Dateisystems entpacken
  - zdb -i, kstat txg_history
- jede synchrone Operation (z. B. fsync(), O_DIRECT, …) würde Commit einer ZFS-Transaktionsgruppe erfordern
  -> Änderung zuzüglich aller ZFS-Metadaten bis hin zu Uberblocks synchron auf Datenträgern
  -> sehr hoher Zusatzaufwand, IOPS-intensiv
- Statt dessen Aufzeichnung synchroner Operationen in Intent Log (ZIL), Umsetzung im Normalfall während nächsten Commits oder Log Replay falls Crash
  -> geringerer Zusatzaufwand, aber nach wie vor IOPS-intensiv
- ZIL entweder in normalem Pool oder auf separatem slog-Device mit hoher IOPS-Leistung
- NFS (ab v3): grundsätzlich synchrones Schreiben beim Schließen einer Datei (commit on close)

Synchrone Operationen: Lessons Learned

- Synchrone Operationen aufgrund ZFS-Struktur sehr aufwändig (IOPS-intensiv)
- NFS-Schreibzugriffe erzwingen standardmäßig synchrone Operationen
- IOPS-optimiertes slog-Device erhöht ZFS-Durchsatz für synchrone Operationen
- NFS-Export von ZFS-Pools ohne hohe IOPS-Leistung (SSD-Pool oder SSD-slog) in der Regel nicht ratsam

Volume Labels, Raid und Storage Pools

- Experimente mit:
  - mehrere Testpools mit unterschiedlicher Struktur (Mirror, Raid-Z, Raid10)
  - zdb -l
  - zdb -R
  - Abbildung von Dateiinhalten auf Platteninhalte beobachten
- Pool aufgebaut aus zweistufiger vdev-Struktur (physisch, logisch)
- Blockgrößen auf vdev zwischen 2^ashift (Sektorgröße) und recordsize (ZFS-Parameter)
- Volume Labels auf jedem physischen vdev enthalten
  - Poolstruktur
  - GUIDs der Komponenten
  - Host-ID
  - Uberblock-Array

Volume Labels: Lessons Learned

- Pool-Import
  - sucht Komponenten anhand Label-Information
  - stellt vdev-Struktur zusammen
- Pool-Export
  - stoppt vdevs
  - gibt Pool für Importe von anderen Geräten (Host-IDs) frei
  - ändert Volume Label (txg)
- Volume-Label
  - einziger Bereich mit fester Position auf Speichergeräten
  - pro Gerät vierfach gespiegelt
- Raid-Z (raidz, raidz2, raidz3)
  - nutzt Transaktionsverhalten von ZFS, stets konsistent
  - nutzt Wissen über logische Blockstrukturen
  - adaptive Stripe-Größen, dadurch Speicher-Overhead für Raid-Gruppen eventuell höher
  - kein Partial-Stripe-Writes

Read-Cache (ARC, L2ARC)

- Experimente mit: arc_summary.py, arcstat.pl
- ZFS implementiert Read-Cache mit eigenem Algorithmus, unabhängig von OS-Cache
  - zweistufig: ARC (RAM), L2ARC (Speicherlaufwerk)
  - teilt Cache adaptiv zwischen zwei Strategien (zuletzt genutzt vs. häufig genutzt)
  - wird beim ersten Lesen befüllt (kein Write-Through/Write-Behind)
  - gedrosselte Füllrate
  - typischerweise lange Anlaufzeiten

Zusammenfassung

- ZFS bietet
  - Speicherverwaltung in redundanten Storage-Pools
  - hierarchische Objektstruktur mit Transaktionssemantik
  - inhärente Integritätssicherung
  - flexible Dateisystemverwaltung
  - skalierbare Snapshot- und Replikationsmechanismen
  - adaptives Cache-Management

Mehr zum Thema

- ZFS RAIDZ stripe width, or: How I Learned to Stop Worrying and Love RAIDZ (Matt Ahrens)
  https://www.delphix.com/blog/delphix-engineering/zfs-raidz-stripe-width-or-how-i-learned-stop-worrying-and-love-raidz

- ZFS Internal Structure (Uli Gräf)
  http://www.osdevcon.org/2009/slides/zfs_internals_uli_graef.pdf

- ZFS On-Disk Data Walk (Or: Where's My Data) (Max Bruning)
  http://www.osdevcon.org/2008/files/osdevcon2008-max.pdf

- ZFS On-Disk Specification
  http://www.giis.co.in/Zfs_ondiskformat.pdf

- Documentation/DnodeSync (OpenZFS)
  http://open-zfs.org/wiki/Documentation/DnodeSync