

Version 1.1, 14.05.07

Das OpenSolaris ZFS Dateisystem

Constantin Gonzalez
constantin.gonzalez@sun.com
blogs.sun.com/constantin

Copyright (C) 2007 Sun Microsystems GmbH.

Dieses Werk kann durch jedermann gemäß den Bestimmungen der Lizenz für die freie Nutzung unveränderter Inhalte genutzt werden.

Die Lizenzbedingungen können unter <http://www.uvm.nrw.de/opencontent> abgerufen oder bei der Geschäftsstelle des Kompetenznetzwerkes Universitätsverbund MultiMedia NRW, Universitätsstraße 11, D-58097 Hagen, schriftlich angefordert werden.

Abstract

Eines der innovativsten Eigenschaften des OpenSolaris Betriebssystems ist das Solaris ZFS Dateisystem.

Es vereint die Aufgabe eines Volume Managers und eines traditionellen Dateisystems in einem und bietet dadurch einen neuen, zuverlässigen, leistungsfähigen und flexiblen Ansatz für die Bereitstellung und Optimierung von Daten-Diensten. Die Copy-on-Write-Semantik und die Verwendung verketteter Checksummen über die ganze Struktur des Dateisystems hinweg liefern eine robuste Konsistenz des Dateisystems (kein fsck(1M) mehr nötig), eine zuverlässige Datenintegrität die den ganzen Datenfluß von der Dateisystem-Schnittstelle bis zum Datenblock auf dem Medium umspannt, und eine Menge von zusätzlichen Eigenschaften, die bisher nur unter großem technischen und finanziellen Aufwand möglich waren.

ZFS ist als Teil von OpenSolaris ein Open Source Dateisystem, das unter einer freien Lizenz verfügbar ist. Die stetig wachsende Gemeinschaft von ZFS-Entwicklern und Anwendern umspannt neben dem OpenSolaris Betriebssystem und seinen Distributionen auch FreeBSD, Mac OS X und Linux (über FUSE).

Dieses Paper gibt eine Einführung in das Solaris ZFS Dateisystem und erklärt Funktionsweise, Architektur und Anwendung dieser Technologie. Es liefert Beispiele von ZFS aus der Praxis und gibt praktische Ratschläge für den Einsatz in eigenen Projekten. Zum Schluss betrachten wir ein paar Neuere Entwicklungen aus der ZFS-Welt, geben einen Ausblick auf zukünftige Projekte rund um ZFS und Hinweise auf weitere Informationen über ZFS.

Warum das Rad neu erfinden?

Traditionelle Dateisysteme haben eine lange Historie, die zum Teil über 20 Jahre in die Vergangenheit reicht, oder darauf aufbauen. Daraus ergeben sich Design-Prinzipien und Annahmen, die nicht mehr den Eigenschaften moderner Speichersysteme entsprechen oder nicht mehr in der Lage sind, heutigen oder zukünftigen Anforderungen an Dateidiensten gewachsen zu sein.

So gehen traditionelle Dateisysteme zum Beispiel davon aus, dass der Datenpfad vom Rechner zur Festplatte zuverlässig sei. Dies ist jedoch in Zeiten komplexer SAN-Infrastrukturen mit zahlreichen Geräten, Verbindungen und anderen Fehlerquellen zwischen Dateisystem auf dem Rechner und Bits auf einer Festplatte längst nicht der Fall, ganz zu Schweigen vom Risiko der Datenmanipulation in verteilten Speicherumgebungen. Auch die weite Verbreitung kostengünstiger Standard-Hardware birgt das Risiko von unerkannter Datenkorruption gegen das traditionelle Dateisysteme keinen Schutz bieten.

Auch bauen viele Dateisysteme auf der Annahme auf, dass Daten in einer Block/Sektor-Struktur auf einem rotierendem, magnetischen Medium mit einem Kopf gespeichert werden und versuchen, die Kommunikation mit und die Speicherung auf so einem Medium zu optimieren. In einer Zeit, in der Flash-Memory-Speicher sowohl für den Heim- als auch für den professionellen Gebrauch weit verbreitet sind und in der selbst eine „normale“ Festplatte mit mehreren Schreib/Lese-Köpfen ihre wahre Geometrie hinter ihrer Firmware versteckt, ist dies nicht mehr opportun, vor allem wenn der Gebrauch von Volume Managern oder RAID-Controllern als zusätzlicher Ebene zwischen Dateisystem und Speichermedium jede Eigenschaft der Speicherhardware für das Dateisystem unsichtbar macht.

Schließlich verwenden traditionelle Dateisysteme 32 oder 64 Bit große Pointer zur Verwaltung ihrer Datenstrukturen, was angesichts des rapide wachsenden Speicherbedarfs in modernen Unternehmen und anderen EDV-Einrichtungen spätestens im nächsten Jahrzehnt an seine Grenzen stoßen wird.

Angesichts dieser und weiterer Nachteile, die sich aus der Historie von Dateisystemen ergeben, haben die OpenSolaris-Entwickler beschlossen, einen fundamental neuen Ansatz zur Verwaltung von Dateien zu entwickeln, der heutigen und zukünftigen Anforderungen an die moderne Speicherung von Daten gerecht wird: ZFS.

Das OpenSolaris ZFS Dateisystem

Einer für alle und alle für einen: Storage Pools

Heutige Speicher-Infrastrukturen bestehen aus mehreren Schichten: Das eigentliche Speichermedium (z.B. Festplatte) wird oft durch einen RAID-Controller oder den Controller eines Speichersubsystems oder auch durch eine Volume Management Software virtualisiert und dem System als ein Block-Device, also eine virtuelle Festplatte dargestellt. Erst auf dieses Block-Device hat das eigentliche Dateisystem Zugriff. Diese Aufteilung in Volume

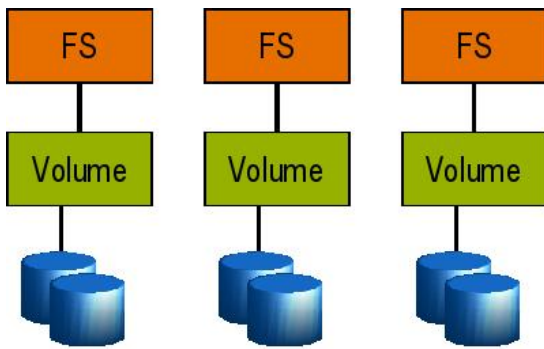


Abb. 1: Traditionelle Aufteilung in Volume Manager und Dateisystem.

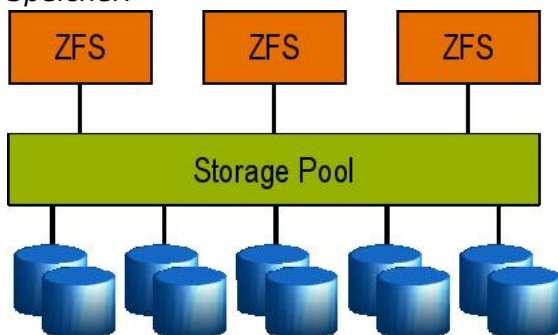
Management- und Dateisystem-Schicht ergibt sich historisch aus der Notwendigkeit, mehrere Festplatten zusammenzufassen, um entweder mehr Kapazität (RAID-0), bessere Verfügbarkeit (RAID-1, RAID-5,6) oder beides zu erreichen.

ZFS kombiniert die Eigenschaften eines Volume Managers und eines Dateisystems durch die Einführung von Storage Pools. Ein Storage Pool fasst mehrere Festplatten zusammen und implementiert eine

Strategie zur redundanten Datenhaltung (z.B. Spiegeln, RAID-Z oder RAID-Z2). Ein oder mehrere ZFS-Dateisysteme bedienen sich dann aus diesem Storage-Pool, in dem sie nur die benötigten Blöcke aus dem Pool entnehmen und für die Speicherung ihrer Daten verwenden.

Während also ein Volume-Manager als Abstraktions-Ebene eine virtuelle Festplatte darstellt, arbeitet ein Storage-Pool eher nach dem Prinzip einer RAM-Speicherverwaltung (malloc/free) und bietet dadurch viele Vorteile:

Abb. 2: ZFS benutzt Storage Pools als fein granulare Abstrahierung von Speicher.



- Flexible, dynamische Zuordnung von Speicherkapazität zu Dateisystemen: Jedes Dateisystem verbraucht nur so viele Speicherblöcke, wie tatsächlich benötigt. Das vergrößern/verkleinern von Dateisystemen ist nicht mehr nötig. Man bekommt eine effizientere Auslastung von vorhandenem Speicherplatz. Der Speicherverbrauch pro ZFS Dateisystem kann vom Administrator durch Quotas

eingeschränkt und/oder eine Mindestgröße reserviert werden.

- Flexiblere Einsatzmöglichkeiten von Dateisystemen: Ein ZFS Dateisystem in einem Pool ist eine Baumstruktur, bestehend aus Speicherblöcken des Pools. Solche Dateisysteme lassen sich sehr schnell erzeugen (< 1 Sek.), sind skalierbar (die Metadaten-Menge wächst harmonisch mit dem verbrauchten Speicherplatz) und können flexibel und einfach verwaltet werden. Beispielsweise ist es nun möglich und sinnvoll, jedem Benutzer eines Systems sein eigenes Dateisystem zur Verfügung zu stellen.
- Bessere Performance: Ein Dateisystem auf einem Volume-Manager ist in seiner Performance auf die Leistung der darunter liegenden Festplatten beschränkt. Ein ZFS-Dateisystem auf einem Storage-Pool kann bei Bedarf die gesamte aggregierte Leistung des darunter liegenden Pools und damit aller Festplatten nutzen (dynamisches Striping).
- Einfache Erweiterung: Wird der Speicherplatz eines Pools zu knapp, kann man ihn einfach um weitere Festplatten erweitern. Die neue Kapazität steht dann sofort allen Dateisystemen zur Verfügung und die Gesamt-Performance des Pools wird durch die zusätzlichen Spindeln erhöht. Als erstes 128-Bit Dateisystem unterstützt ZFS eine Datenkapazität von 2^{128}

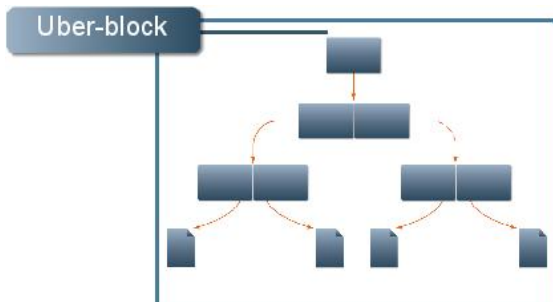
Byte – genug Platz, um bei Verwendung heutiger Festplatten allein durch deren Stromaufnahme die Ozeane dieser Erde verdampfen zu lassen¹!

ZFS Storage-Pools sind also für die Verwaltung von Speicherressourcen auf Hardware-Ebene verantwortlich. Auf dieser Ebene implementiert ZFS auch Spiegelung (RAID-1) oder andere Redundanz-Mechanismen (RAID-Z bzw. RAID-Z2, siehe weiter unten). Dabei werden zwei gespiegelte Festplatten oder eine RAID-Gruppe in einem virtuellen Device („vdev“) zusammengefasst. Mehrere vdevs, nach RAID-0-Manier gekoppelt, stellen dann die Speicherkapazität für einen Pool zur Verfügung. Vdevs können neben Spiegel- oder RAID-Z/Z2-Gruppen auch einzelne Platten, Partitionen (Slices) oder auch ganz normale Dateien (z.B. für Demo- oder Test-Zwecke) sein.

Sicher ist sicher: Copy-on-write-Transaktionen

Wie vorhin erwähnt ist ein ZFS Dateisystem eine Baum-artige Datenstruktur,

Abb. 3: ZFS Block-Struktur. Die Blätter des Baumes sind Datenblöcke.



bestehend aus Metadatenblöcken und Nutzerdatenblöcken aus dem Storage-Pool. Die Wurzel ist dabei der sog. „Überblock“ (vom englischen aus dem deutschen übernommen, aber ohne Ü-Punkte), der Zeiger auf darunter liegende Metadatenblöcke enthält. Metadatenblöcke wiederum enthalten weitere Zeiger (bis zu 256) auf darunterliegende Metadatenblöcke oder Datenblöcke. Dateisystemdaten aber auch Verzeichnisstrukturen, ACLs etc. sind in Datenblöcken gespeichert.

Sollen Daten im ZFS Dateisystem verändert werden, arbeitet ZFS streng

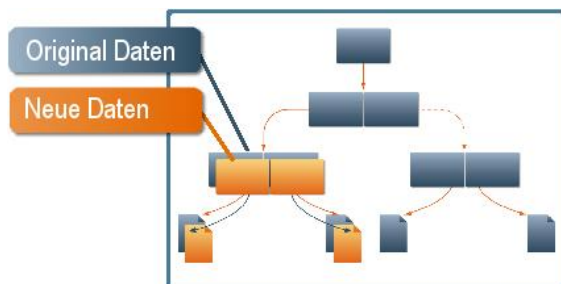


Abb. 4: Anlegen neuer Daten- und Metadatenblöcke.

transaktional: Es werden nie aktive Blöcke überschrieben, sondern stets nur neue Blöcke angefordert. In Abb. 3-6 wird dies anhand eines Beispiels dargestellt. Sollen z.B. die unteren linken zwei Datenblöcke verändert werden, so wird zuerst der neue Inhalt in zwei neue Datenblöcke (darüberliegend, orange dargestellt) abgespeichert.

Um die neuen Datenblöcke zu referenzieren, muss der dazugehörige Metadatenblock verändert werden, was auch hier zunächst durch Anlegen eines neuen Metadatenblocks mit dem neuen Inhalt (Pointer auf die veränderten Datenblöcke) geschieht. Dies setzt sich fort, bis zum Schluss ein neuer Überblock angelegt wird. Dieser bekommt eine höhere Seriennummer als der alte Überblock, was ihn als den neuen aktiven Überblock auszeichnet.

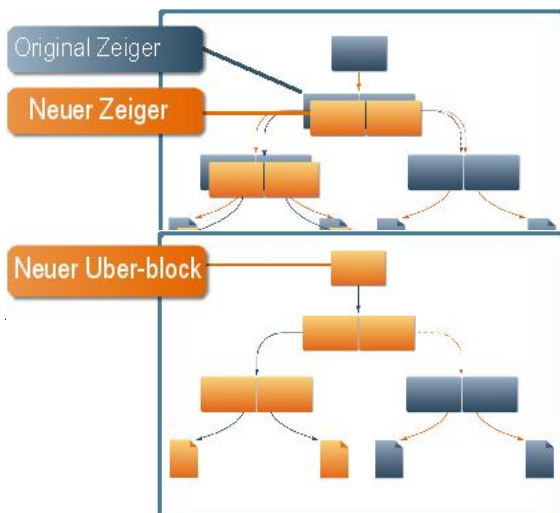


Abb. 6: Ein neu angelegter Überblock markiert das erfolgreiche Ende einer Transaktion.

Erst wenn der neue Überblock erfolgreich auf die Festplatte geschrieben wurde, gilt diese Transaktion als abgeschlossen (wie ein „Commit“ in einer Datenbank) und damit die Datei als erfolgreich verändert.

Durch diese Strategie, die stark an Datenbank-Transaktionen angelehnt ist, ist der Zustand eines ZFS Dateisystem jederzeit konsistent, auch bei Systemausfall oder anderen Hardware-Fehlern. Dadurch ist eine langwierige Überprüfung der Dateisystemstruktur durch fsck(1M) nach einem Systemausfall nicht mehr nötig, denn das ZFS Dateisystem befindet sich logisch gesehen zu jeder Zeit in einem gültigen und korrekten Zustand.

Alte Überblöcke werden zur Sicherheit noch eine Zeitlang aufbewahrt und seit Version 2 des ZFS On-Disk-Formats werden alle Metadatenblöcke (inkl. Überblöcke) bis zu dreifach repliziert (sog. Ditto-Blocks²) in einem Pool vorgehalten, um bei etwaiger, hardwarebedingter Korruption einen zusätzlichen Schutz der Metadaten zu gewährleisten.

Frei gewordene Blöcke (also die, auf die nach erfolgreichen Transaktionen nicht mehr referenziert wird) werden in einer Liste von freien Blöcken verwaltet und können dann wieder verwendet werden.

Copy-on-Write hat auch einen Geschwindigkeits-Vorteil: Da beim Schreiben von Daten keine alten Blöcke mehr überschrieben werden, müssen die Schreib/Lese-Köpfe der Festplatten auch nicht mehr deren Position anfahren. Stattdessen werden ja nur freie Blöcke im Pool verwendet, wobei ZFS diese so auswählen kann, dass die Festplattenköpfe möglichst lange hintereinander liegende Reihen von Blöcken auf der Platte beschreiben können („Sequential Writes“). Dies ist aus physikalischen Gründen deutlich schneller, als zufällig auf der Platte verteilte Blöcke anzufahren und zu überschreiben („Random Writes“). ZFS kann also zufällige Schreib-Lasten, die bei traditionellen Dateisystemen zu Performance-Problemen führen, geschickt in sequenzielle Schreib-Lasten umwandeln, die weit schneller ausführbar sind.

Bitte lächeln: Snapshots

Aus der oben beschriebenen Copy-on-Write-Semantik ergibt sich ein einfacher, eleganter und Platz sparender Weg, einen Schnappschuss (Snapshot) des gesamten Dateisystems zu machen: Man behält einfach den alten Überblock und verwendet ihn als Einstieg in die Struktur, die er referenziert, das Snapshot-Dateisystem.

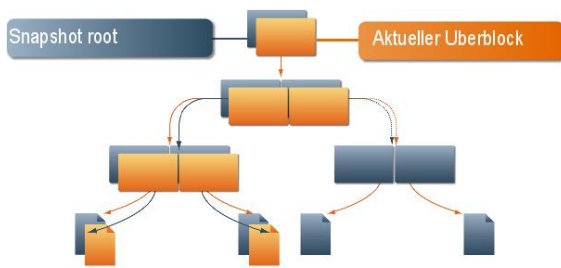


Abb. 7: Ein Snapshot entsteht durch Beibehaltung eines alten Überblocks.

Snapshots sind zunächst also nur-lesemarkierte, auf alten Überblöcken beruhende ZFS-Dateisysteme. Diese teilen sich Blöcke mit dem entsprechenden aktiven Dateisystem. Erst wenn nach der Erzeugung des Snapshots das aktive Dateisystem verändert wird, fällt für den Snapshot ein Speicheranforderung an (nämlich die ersetzten Blöcke, die nun nicht mehr freigegeben werden). In Abb. 7

wird der Snapshot in blau (darunterliegend) und das aktive Dateisystem in orange (darüberliegend) dargestellt. Der rechte untere Teilbaum wird sowohl vom Snapshot als auch vom aktiven Dateisystem benutzt, die orangefarbenen Bereiche sind bei beiden Dateisystemen unterschiedlich.

Snapshots können in Clones umgewandelt werden, d.h., sie werden wieder schreibbar gemacht. Auch hier entsteht durch die gemeinsam verwendeten Blöcke ein Einspareffekt. Erst wenn ein Clone befördert wird („Clone Promotion“), wird er von seiner Herkunft unabhängig gemacht und der damit assoziierte Snapshot bzw. das Mutter-Dateisystem kann gelöscht werden.

Snapshots in ZFS verbrauchen also minimalen Speicherplatz. Sie können in Sekunden und mit wenig Aufwand angelegt werden und stellen daher ein mächtiges und effizientes Werkzeug für den Systemadministrator dar.

Vertrauen ist gut – Kontrolle ist besser: Daten-Integrität durch und durch

Moderne Speichersysteme versuchen, sich durch Replikation von Daten (Mirroring) oder durch Paritäts-Verfahren (RAID-5, RAID-6) vor Datenverlust zu schützen. Hierbei wird jedoch vorausgesetzt, dass der Datenpfad vom Server bis zum RAID-Controller bzw. zur Festplatte zuverlässig arbeitet. Auch wird darauf vertraut, dass ein Speichermedium beim Lesen eines Datenblocks zuverlässig erkennen kann, ob dieser korrekt oder fehlerhaft gelesen werden konnte, was moderne Festplatten i.A. durch in den Datenblöcken eingebettete Checksummen versuchen, zu erreichen.

Was passiert jedoch, wenn Daten bereits auf dem Wege vom Rechner über eine Schnittstellenkarte (Firmware/HW-Fehler...), einen Datenpfad (USB, eSATA, FireWire, Fiberchannel, etc.: Interferenzen, Überspannungen...), einen Storage-Controller (FW/HW-Fehler...), den darauf folgenden Datenpfaden (s.o.) und schließlich dem Festplattencontroller selbst (auch hier: FW-Fehler, Überspannungen...), willentlich oder unbeabsichtigt verändert oder korrumpiert werden? Was wenn der Festplattenkopf beim Schreiben einen Positionierungs-Fehler macht und einen Datenblock zwischen zwei Spuren, an die falsche Stelle oder gar über einen bestehenden Datenblock schreibt?

Alle diese Fehlerszenarien können nicht durch traditionelle Block-Checksummen einer Festplatte abgesichert werden, dennoch treten sie in der Realität auf und sie werden angesichts stetig steigender Datenmengen in Zukunft häufiger auftreten. In der Praxis machen sich solche unerkannten Datenkorruptions-Fehler durch ungeklärte System-Abstürze (Z.B. bekommt ein

Dateisystem einen Metadatenblock zurück, mit dem es nicht gerechnet hatte und löst eine Panic aus, oder es wird ein Kernel-Modul geladen, dessen Code korrumpiert ist und einen Absturz verursacht), inkonsistente Zustände von Spiegeln (beide Seiten „behaupten“ was anderes, beide denken, sie haben „Recht“) oder schlicht und ergreifend falsche Daten („Der Kontostand beträgt -1,2089258196146e24 Euro“) bemerkbar, wenn es schon zu spät ist.

So ist z.B. Anfang des Jahres ein Mail-Server mit ca. 600GB Mail-Daten aufgrund von korrupten Daten in den UFS-Inodes abgestürzt. Beim wieder hochfahren dauerte der fsck(1M)-Lauf ca. 10-12 Stunden, doch er führte nicht zum Erfolg, da beide Spiegelhälften verschiedene Daten enthielten, was weitere Abstürze zur Folge hatte. Die Ursache für dieses zunächst unerklärliche Verhalten war ein defekter Fiber-Channel-Controller, der bei der Übertragung von Daten ins SAN Fehler in den Datenstrom injizierte.

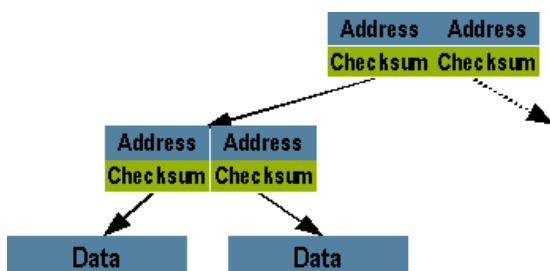


Abb. 8: Verkettete Checksummen in ZFS.

ZFS verwendet daher bereits auf der Dateisystem-Ebene Checksummen für alle Daten- und Metadatenblöcke. Dies ist angesichts heute verfügbarer CPU-Kapazitäten kein wesentlicher Aufwand, sichert die Daten jedoch bereits auf dem Server wirksam vor Datenkorruption.

Dabei geht ZFS gegenüber traditionellen Block-Checksummen einen Schritt weiter:

Die Checksumme für einen Datenblock wird im darüberliegenden Metadatenblock gespeichert. Auch die Checksumme eines Metadatenblocks wird eine Ebene höher durch den darüberliegenden Metadatenblock gespeichert. Schließlich enthält der Überblock alle Checksummen für die erste Ebene von Metadatenblöcken. Durch diese verkettete Struktur von Checksummen über die gesamte ZFS-Dateisystem-Struktur hinweg wird gewährleistet, dass zu jedem Zeitpunkt und über jede Art von Storage-Infrastruktur hinweg Datenkorruption zuverlässig erkannt werden kann.

Der Algorithmus für die Checksummen kann gewählt werden. Zur Zeit ist der Algorithmus „fletcher2“ voreingestellt, der eine gute Performance bei guter Fehlerabsicherung bietet. Wer eine härtere Absicherung vor Datenkorruption möchte, kann einen anderen Algorithmus wählen, etwa „SHA256“.

Durch dieses Verfahren entspricht am Ende die Checksumme des Überblocks einer digitalen Signatur, die das gesamte Dateisystem gegen unbemerkte Veränderung absichert³.

Aus dieser robusten und durchgängigen Datenintegrität ergibt sich ein weiteres Vorteil der bei gespiegelten oder durch andere RAID-Verfahren mit Redundanz versehenen Daten zum Tragen kommt: Liest ZFS aus der einen Spiegelhälfte einen Block und stellt dann aufgrund der verketteten Checksummen fest, dass dieser falsche Daten enthält, kann ZFS den gleichen Block versuchen, von der anderen Spiegelhälfte zu lesen. Ist der gespiegelte Block korrekt, so kann ZFS den korrekten Block an die Applikation weitergeben, und dabei den beschädigten Block auf der ersten Spiegelhälfte korrigieren.

Ein normaler Volume-Manager oder ein normales Storage-Subsystem hätte in diesem Beispiel trotz Mirroring oder RAID-Technologien den falschen Block unbemerkt an die Applikation weitergegeben, siehe unser eingangs erwähntes

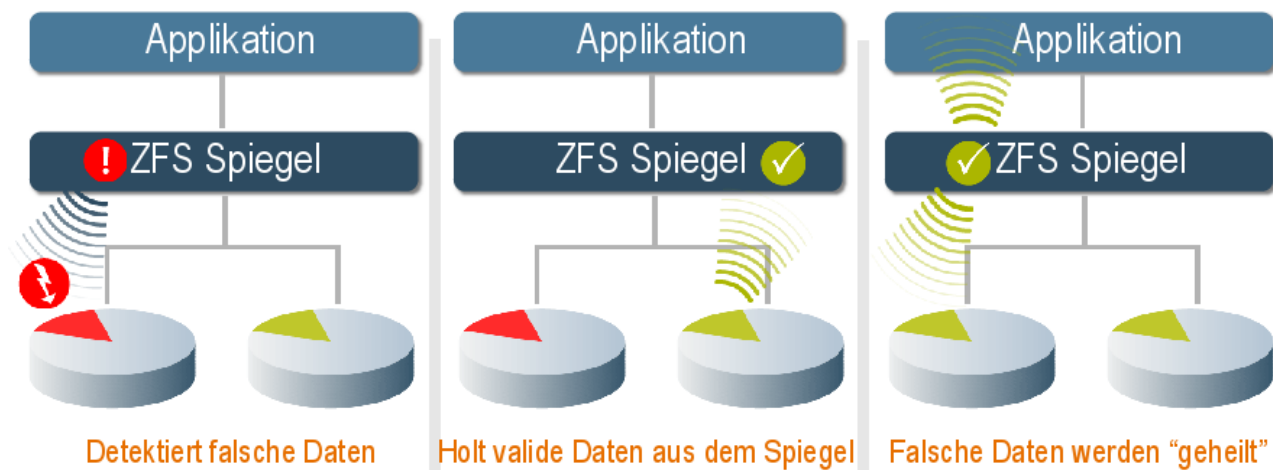


Abb. 9: Automatische Datenreparatur in ZFS-Spiegeln.

Beispiel aus der Praxis mit dem Mail-Server.

Um stiller Datenkorruption durch sich auf alternden Medien verändernder Bits vorzubeugen bietet ZFS eine „Scrubbing“-Funktion. Einmal angestoßen, liest dann ZFS rekursiv jeden aktiven Block eines Dateisystems und repariert ihn bei Bedarf mit Hilfe vorhandener Redundanz (Es ist also nötig, den Pool aus gespiegelten oder mit RAID-Z/Z2 abgesicherten Geräten aufzubauen) automatisch. Nach Abschluss des Scrub-Vorgangs erhält der Administrator eine Statistik über evtl. fehlgeschlagene Lese-Vorgänge und der Information, ob die zugrunde liegenden Daten erfolgreich repariert werden konnte.

Auf dem Heim-Server des Autors ist es schon vorgekommen, dass ZFS einen korrupten Block erkannt und erfolgreich repariert hat⁴. Dies wäre bei einem traditionellen Dateisystem nicht aufgefallen, sondern hätte mit der Zeit zu stiller Datenkorruption geführt. Daher ist die regelmäßige Verwendung des ZFS-Scrubbers (etwa über die `crontab(1)` bei preisgünstigen Platten etwa einmal die Woche und bei professionellen Platten etwa einmal im Monat sehr zu empfehlen.

Fünf Freunde zum Quadrat: RAID-Z und RAID-Z2

Bisher haben wir in unseren Beispielen gespiegelte virtuelle Geräte („vdev“) betrachtet. ZFS unterstützt auch eine Variante des RAID-5 Algorithmus, das sog. „RAID-Z“. Bei traditionellem RAID-5 werden Datenblöcke auf n Festplatten aufgeteilt, wobei die $n+1$. Festplatte für Paritäts-Checksummen benutzt wird. Fällt eine der $n+1$ Festplatten aus, können die Daten durch Kombination der Daten aus den Überlebenden Daten wieder rekonstruiert werden.

Hierbei besteht jedoch die Gefahr des sog. „RAID-5 Write-Hole“: Beim Speichern neuer Daten werden auf allen $n+1$ Festplatten parallel Schreibvorgänge angestoßen. Fällt das Festplatten-System (z.B. Stromausfall) aus, während nicht alle Schreibvorgänge abgeschlossen sind, ist der Datenblock korrupt. Aus diesem Grund verwenden moderne Speichersysteme und RAID-Controller einen batteriegepufferten Cache, der jedoch mitunter viel Geld kostet.

Durch die Kombination aus dem RAID-5 Algorithmus und der ZFS Copy-on-Write-Semantik ergibt sich hier ein Vorteil: Jetzt kann ein Ausfall der

Festplatten mitten im Schreibvorgang nicht mehr zu einer Inkonsistenz führen, denn der Vorgang gilt erst dann als abgeschlossen, wenn alle Teilvorgänge auf allen Festplatten erfolgreich beendet werden konnten. Nach einem Ausfall des Festplatten-Systems kann ZFS anhand der vorhandenen (evtl. älteren) Überblöcke leicht feststellen, welches der neueste, konsistente Zustand des Dateisystems ist und diesen dann weiter verwenden.

Bereits oben haben wir betrachtet, wie der Copy-on-Write-Mechanismus dafür sorgt, dass zufällige Schreib-Vorgänge in sequenzielle Schreibvorgänge für die Festplatten umgewandelt werden. Bei RAID-Z kommt ein weiterer Vorteil hinzu: Der traditionelle Zyklus aus Lesen des alten Blocks, Modifizieren des Blocks inkl. Generierung von neuer Parity-Information, Rückschreiben des neuen Blocks, also ein drei-Schritt-Prozess, reduziert sich jetzt auf den reinen Schreib-Vorgang, da wie gesagt nur freie Blöcke beim Schreiben benutzt werden. Dies reduziert die Anzahl der I/O-Vorgänge auf Festplatten-Seite pro Schreibvorgang auf die Hälfte.

ZFS verwendet standard-mäßig dynamische Block-Größen, die dem Verhalten der Anwendung angepasst werden. Bei RAID-Z mit $n+1$ Platten kann ZFS je nach Belastung durch die Anwendung pro Schreibvorgang die Daten auch auf weniger als n Platten verteilen, um so Platz zu sparen oder die Performance zu optimieren.

RAID-Z2 erweitert das RAID-Z-Konzept, ähnlich RAID-6 um eine zusätzliche Festplatte für weitere Parity-Informationen ($n+2$ Festplatten). Hierdurch kann der Ausfall von 2 Festplatten gleichzeitig toleriert werden, was vor allem bei Szenarien wichtig ist, in denen eine Festplatte schon ausgefallen ist, die Ersatz-Festplatte jedoch noch nicht vollständig im RAID-Verbund integriert wurde und daher dar RAID-Satz verwundbar ist.

RAID-1, RAID-Z und RAID-Z2 sind in ZFS nicht nur robustere Implementationen von RAID-Algorithmen, sie bieten darüber hinaus auch automatische Datenreparatur und bessere Performance, so dass oft die Investition in aufwändige RAID-Controller eingespart werden kann⁵.

Fällt bei einem ZFS-Spiegel oder RAID-Z/Z2 Satz eine Festplatte aus und ist eine oder mehrere Hot-Spare Platte konfiguriert, so ersetzt ZFS automatisch die defekte Platte durch die neue Platte und fängt dann an, diese mit den (rekonstruierten) Daten der ersetzten Platte zu füllen. Dieser Vorgang heißt „Resilvering“ und läuft mit ZFS deutlich effizienter ab, als bei einem traditionellen RAID-System: Dadurch, dass das Dateisystem und der Volume-Manager im ZFS Pool-Konzept eine Einheit darstellen, weiß der Pool genau, welche Blöcke auf der Ersatz-Festplatte repliziert werden müssen und welche nicht. Dies kann die Wiederaufbauzeit bei nicht ausgelasteten Dateisystemen entscheidend verkürzen, da nur so viele Daten wie nötig repliziert werden.

Für die Ewigkeit gebaut: Die ZFS-Architektur

Bisher haben wir die wichtigsten Eigenschaften von ZFS besprochen und schon ein Gefühl dafür bekommen, wie es funktioniert. Im Folgenden schauen wir uns kurz die ZFS-Architektur an, und wie die o.a. Technologien darin zusammengefügt sind.

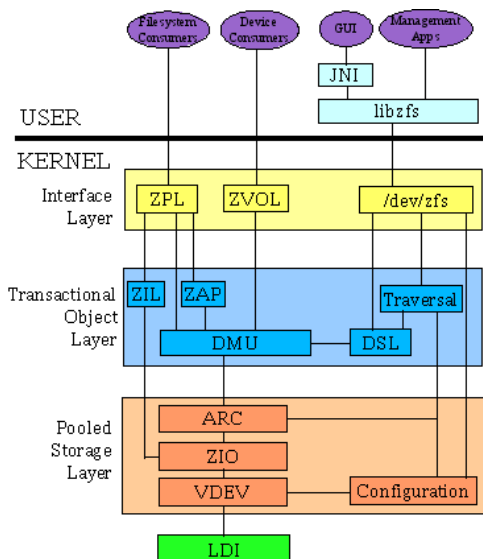


Abb. 10: Die ZFS Architektur

Abb. 10 Zeigt den Aufbau von ZFS und gliedert seine Moduln in 3 Schichten auf.

- Im „Pooled Storage Layer“ sind die Komponenten zusammengefasst, die Hardware-nah aus einem oder mehreren Speichergeräten die Pool-Abstraktion von ZFS herstellen. Das VDEV-System interagiert mit Geräten über das „Layered Driver Interface“ (LDI) und stellt eine Baumstruktur aus Vdevs zur Verfügung, wobei die Blätter dieses Baums die eigentlichen Geräte sind und die darüberliegende Ebene für Spiegel, RAID-Z und RAID-Z2 Vdevs zuständig ist.

Zusammen mit dem Pool Configuration-Modul ergibt sich die Basis-Funktionalität der ZFS-Pools.

Die „ZFS I/O-Pipeline“ (ZIO) implementiert den Datenfluß zu und vom ZFS Pool, inklusive Kompression und Checksummen-Behandlung, Aufteilung von Daten in Blocks und die effiziente Abwicklung von I/O über Vdevs. Schließlich stellt der „Adaptive Replacement Cache“ (ARC) die primäre Schnittstelle für die nächste Ebene zur Verfügung und verwaltet die ZFS-Cache-Infrastruktur.

Der Pooled Storage Layer wird oft vereinfacht auch „Storage Pool Allocator“ (SPA) genannt.

- Nach der Hardware-Abstraktion durch den SPA finden im „Transactional Object Layer“ alle höherwertigen Funktionen von ZFS statt. Die „Data Management Unit“ (DMU) macht aus dem flachen Adressraum des SPA eine Objekt-basierte und Transaktions-orientierte Speicher-Infrastruktur und ist damit u.a. auch für die Konsistenz der Daten auf dem Medium verantwortlich. Dabei hilft ihm der „Dataset and Snapshot Layer“ (DSL), der DMU-Objekte in einem hierarchischen Namensraum mit vererbzbaren Properties organisiert, Quotas und Reservierungen durchsetzt und Snapshots und Clones verwaltet. Ein weiteres Hilfsmittel für die DMU ist der „ZFS Attribute Processor“ (ZAP), der eine skalierbare, Hash-orientierte Verwaltung von Name-Objekt-Tupeln implementiert. Diese werden dann für die effiziente Verwaltung großer Mengen von Verzeichnis- und Dateinamen genutzt sowie für andere Aufgaben des DSL. Das „ZFS Intent Log“ (ZIL) puffert die Bedürfnisse von Applikationen ab, die ein synchrones Schreiben von Daten auf die Hardware verlangen (O_DSYNC), während der SPA sonst aus Performance-Gründen mehrere I/Os zu größeren Transaktionsgruppen aggregiert, die periodisch auf das Medium geschrieben werden. Schließlich liefert das Traversal-Modul einen sicheren und schnellen Weg, alle Daten in einem Pool zu durchlaufen, um die Scrubbing bzw. die Resilvering-Funktionalität in optimaler Zeit durchzuführen.
- Die Interaktion zwischen ZFS im Kernel und seinen Verbrauchern im

Userspace wird durch das „ZFS Interface Layer“ realisiert.

Die „ZFS POSIX Layer“ Schicht implementiert die POSIX-gerechte Dateisystem-Semantik und ist auch für die Behandlung von ACLs (nach NFS v4 bzw. NT Art) zuständig. Sie interagiert mit dem OpenSolaris „Virtual File System“ Interface.

ZFS kann aber auch für andere Zwecke genutzt werden, so sitzt hier auch das ZVOL-Modul, das Block-Devices emulieren kann, so daß man ZFS mit allen seinen Vorteilen auch als Volume-Manager für andere Dateisysteme oder andere Nutzer von Block-Devices (z.B. Datenbanken) verwenden kann. Auch das NFS-Modul befindet sich in dieser Schicht, ist aber hier nicht extra eingezeichnet.

Die Verwaltung von ZFS geschieht über das /dev/zfs-Device, an das im Userspace die libzfs-Bibliothek anknüpft, die von den Befehlen `zpool(1M)`, `zfs(1M)`, dem Web-basierten GUI (über JNI) oder potenziell anderen Management-Werkzeugen benutzt werden.

Der modulare Aufbau von ZFS erlaubt eine einfache und flexible Erweiterung und Wartbarkeit seiner Eigenschaften und sorgt damit für die Zukunfts-Sicherheit und Entwickler-Freundlichkeit dieser Technologie. Eine detaillierte Beschreibung der ZFS-Architektur inkl. Besprechung des reich kommentierten Source Codes findet sich auf OpenSolaris.org⁶.

Jetzt aber ran: ZFS in der Praxis

Wir haben nun ZFS in der Theorie kennen gelernt, doch am besten verstehen wir seine Möglichkeiten, wenn wir es selber einmal ausprobieren⁷:

Erste Schritte

ZFS hat auch das Ziel, die Administration von Dateisystemen wesentlich zu vereinfachen. Durch die historisch bedingte Aufteilung in Volume Manager und Dateisystem (evtl. mit zusätzlicher RAID-System-Administration) und durch das aufwändige und schlecht skalierende Anlegen von Metadaten-Strukturen bei der Erzeugung von ufs-Dateisystemen ist der Umgang mit traditionellen Dateisystemen in der Praxis komplex, langwierig und fehleranfällig geworden.

Ein ZFS-Pool ist dagegen sehr leicht und schnell angelegt. Der Befehl:

```
zpool create meinpool c0t0d0
```

legt einen neuen Pool mit dem Namen `meinpool` an und benutzt dabei das Gerät `c0t0d0`. Er legt dabei auch automatisch das erste ZFS-Dateisystem in diesem Pool an, das ebenfalls `meinpool` heißt und hängt ihn automatisch im `root`-Dateisystem des Rechners an. Der Befehl dauert etwa eine Sekunde, danach können wir erste Daten nach `/meinpool` kopieren. Der Mount-Punkt `/meinpool` ist bereits im Pool vermerkt, so dass die Konfiguration auch nach einem Reboot gültig ist, ohne eine Konfigurations-Datei verändern zu müssen.

Das Anlegen eines Pools mit gespiegelten Platten ist ebenso einfach:

```
zpool create meinspiegel mirror c0t0d0 c1t0d0
```

erzeugt einen neuen Pool namens `meinspiegel`, der die beiden Geräte `c0t0d0` und `c1t0d0` spiegelt. Ähnlich funktioniert das Einrichten eines pools aus

mehreren vdevs oder eines RAID-Z oder RAID-Z2-Pools:

```
zpool create grosserpool raidz c0t0d0 c0t1d0 c0t2d0 c0t3d0 \  
raidz c1t0d0 c1t1d0 c1t2d0 c1t3d0
```

ZFS Dateisysteme können hierarchisch wie Verzeichnisse in einem Dateibaum verschachtelt werden. Legen wir zum Beispiel ein paar Verzeichnisse für mehrere User auf unserem Spiegel meinspiegel an:

```
zfs create meinspiegel/home  
zfs create meinspiegel/home/ernie  
zfs create meinspiegel/home/bert  
zfs create meinspiegel/home/samson
```

Properties

Wir wollen einen anderen Mount-Punkt haben, denn die User-Verzeichnisse sollen lieber unter /export auftauchen:

```
zfs set mountpoint=/export meinspiegel/home
```

Das genügt, um alle drei Home-Verzeichnisse fortan unter /export/home erscheinen zu lassen. Der Vorteil von hierarchischen Dateisystemen in ZFS ist, dass durch Vererbung von Eigenschaften selbst die Administration von sehr vielen Dateisystemen einfach zu handhaben ist.

mountpoint ist eine „Property“ des ZFS Dateisystems home, das zum Pool meinspiegel gehört. Properties lassen sich mit `zfs get/set` anzeigen bzw. ändern, und zwar zur Laufzeit und mit sofortiger Wirkung. Weitere Properties sind: `compression`, `quota`, `reservation`, `setuid`, etc. Sie sind meist selbsterklärend. Eine besonders praktische Property ist `sharenfs`. Mit:

```
zfs set sharenfs=rw meinspiegel/home
```

sind automatisch alle Home-Verzeichnisse unserer User per NFS verfügbar. Auch hier müssen wir dazu keine Konfigurations-Datei anfassen und falls der NFS-Server noch nicht lief, erledigt ZFS auch das Starten von NFS für uns.

Snapshots

Wir wollen nun einen Snapshot für unseren User Ernie anlegen, denn er möchte demnächst wesentliche Teile seines Projektes verändern:

```
zfs snapshot meinspiegel/home/ernie@Montag
```

Das @-Zeichen weist also auf einen Snapshot hin, der Name des Snapshots folgt danach und ist frei wählbar. Ernie kommt am Mittwoch in unser Büro und hat ein Problem: Er hat aus versehen eine wichtige Datei gelöscht! Zum Glück haben wir einen Snapshot von Montag, als die Datei noch existierte. Wir könnten nun den Zustand von Montag einfach mit:

```
zfs rollback meinspiegel/home/ernie@Montag
```

wieder herstellen. Doch Ernie hängt aber auch an anderen Dateien, die er in der Zwischenzeit verändert hat. Daher ist es besser, ihm einfach Zugriff auf die Daten des Snapshots zu geben. Mit:

```
zfs set snapdir=visible meinspiegel/home
```

werden alle Snapshots aller Verzeichnisse unter und einschließlich home

sichtbar. Im Falle von Ernie, kann er seine verlorene Datei unter:

```
/export/home/ernie/.zfs/snapshot/Montag/Kuchen.c
```

wieder bekommen.

Replizierung

ZFS Snapshots lassen sich in serielle Datenströme verwandeln. Dies ist z.B. für Backup-Zwecke nützlich. So können wir z.B. Berts Home-Verzeichnis mit all seinen Inhalten nach einem Snapshot in eine Datei schreiben:

```
zfs snapshot meinspiegel/home/bert@Backup_vom_Montag
zfs send meinspiegel/home/bert@Backup_vom_Montag \
> /export/backups/berts_backup.zfs
```

Das Gegenstück, dazu heißt `zfs receive`.

Mit einer Kombination aus beidem und `ssh` läßt sich ein ganzes Dateisystem über ein Netzwerk von einem Rechner auf ein anderes replizieren:

```
zfs send meinspiegel/home/bert@Backup_vom_Montag \
| ssh rz2.sesamstrasse.de zfs receive spiegel2/bert
```

Hat man mehrere Snapshots desselben Dateisystems, kann `zfs send` auch inkrementell arbeiten und nur die Blöcke liefern, die zwischen zwei Snapshots unterschiedlich sind:

```
zfs send -i meinspiegel/home/bert@Backup_vom_Montag \
meinspiegel/home/bert@Backup_vom_Dienstag | ssh ...
```

Wir sehen: ZFS läßt sich sehr einfach und intuitiv von der Kommandozeile aus bedienen und auch ein Web-Basiertes GUI ist für ZFS verfügbar. Es reicht, die beiden Befehle `zpool(1M)` und `zfs(1M)` in den man-Pages nachzuschlagen, um in kürzester Zeit die ersten Schritte mit ZFS zu machen.

Tipps

- Redundanz einplanen: ZFS kann seine Stärken erst dann voll ausspielen, wenn Daten redundant vorgehalten werden können, also der Pool entweder aus gespiegelten oder mit RAID-Z/RAID-Z2 abgesicherten Vdevs aufgebaut ist.
Wann soll man was nehmen? Hier gilt die alte Regel der Dreifaltigkeit von Speicher-Eigenschaften: Geschwindigkeit, Zuverlässigkeit und Kapazität. Zwei davon können wir uns aussuchen. Wenn wir davon ausgehen, dass Zuverlässigkeit eine Priorität darstellt, dann bleibt uns die Wahl, ob wir lieber einen schnellen Pool haben und dafür Kapazität opfern wollen, oder lieber einen großen Pool, der nicht immer der schnellste ist. Ersteres erreichen wir durch Mirroring, Letzteres durch RAID-Z bzw. RAID-Z2. Ab fünf Platten pro Vdev ist es sinnvoll, darüber nachzudenken statt RAID-Z lieber RAID-Z2 zu nehmen und von der besseren Verfügbarkeit dieser Variante zu profitieren^{8,9}.
- Wann nehme ich Pools und wann Dateisysteme?
Am einfachsten ist es, Pools als einen Weg zu betrachten, Speicher-

Hardware und ihre Eigenschaften zu organisieren, während man über die ZFS-Dateisysteme die Präsentation dieses Speichers gegenüber Applikationen und Benutzern verwalten kann.

Daher ist es oft sinnvoll, pro Speicherkategorie einen eigenen Pool bereitzustellen, etwa ein Pool, der schnelles, teures und gespiegeltes Storage liefert, ein weiterer mit günstigeren, aber langsamen Platten in RAID-Z-Manier und vielleicht ein dritter Pool, der ein paar alte Festplatten für temporäre Zwecke bereitstellt.

In SANs kann es nützlich sein, Pools als Einheiten zu betrachten, die bei Bedarf schnell von einem System exportiert und in ein anderes importiert werden können (z.B. in einem Failover-Szenario). Hier hilft der Befehl `zpool export/import`, wobei ZFS-Pools sich automatisch an „Little Endian“ bzw. „Big Endian“ Architekturen anpassen (ZFS kann auf beiden Architekturen beide Varianten lesen und schreibt grundsätzlich neue Daten in der nativen Variante), also über beliebige Systemarchitekturen hinweg verwendet werden können.

Durch die Verwendung hierarchischer Dateisysteme auf einem Pool kann man dann wie oben beschrieben sehr schnell und einfach auch große Mengen von Dateisystemen (pro User, pro Anwendung, pro Buchstaben im Alphabet für Mailfolder, etc.) verwalten¹⁰.

- Snapshots sind unsere Freunde!

ZFS Snapshots sind schnell, einfach und kosten wenig. Speicher-Hardware kostet immer weniger, wogegen Datenverlust sehr teuer sein kann. Daher ist es ratsam, oft und großzügig Snapshots zu nutzen. Zum Beispiel per crontab-Eintrag. Tim Foster hat einen Solaris SMF-Service geschrieben, der für beliebige Dateisysteme nach bestimmten Regeln automatisch Snapshots erzeugt und bei Bedarf auch wieder löscht¹¹. Chris Gerhard läßt automatisch jede Minute einen Snapshot pro Dateisystem erzeugen und hat in kurzer Zeit über 60000 davon angehäuft. Er fragte sich neulich, wieviel zusätzlichen Speicher-Aufwand seine Snapshots wohl erzeugt haben und kam dabei nur auf etwas über 2 Prozent (2,13 GB Daten für Snapshots vs. 91,2 GB Daten in seinen Dateisystemen)¹². Er schrieb auch ein Script, das bei jedem Einloggen eines Samba-Benutzers automatisch einen Snapshot für ihn erzeugt¹³. Im Zweifel sind Snapshots also immer eine gute Idee.

- Integration von ZFS mit anderen OpenSolaris-Funktionen:

- Wie oben erwähnt, spielt ZFS automatisch mit der Service Management Facility (SMF¹⁴) zusammen, wenn z.B. die `share nfs` Property verwendet wird. Auch Tims Snapshot-Service¹⁵ ist ein gutes Beispiel für SMF-Integration.
- ZFS unterstützt neben POSIX-Dateisystemen auch das Erzeugen eines „Zvol“. Das ist ein ganz normales Block-Device (ZFS Volume), das auf der Infrastruktur von ZFS aufbaut und für andere Dateisysteme verwendet werden kann. Hier ist ZFS elegant in das OpenSolaris iSCSI-Framework integriert. Die Befehle:

```
zfs create -V 50g meinspiegel/meinvolume
zfs set shareiscsi=on meinspiegel/meinvolume
```

erzeugen ein 50 GB großes Block-Device im Pool `meinspiegel` und exportieren es über iSCSI. Ähnlich wie bei `share nfs` wird auch hier automatisch der iSCSI Target-Daemon über SMF gestartet.

Ben Rockwood beleuchtet dies in seinem Blog¹⁶ und zeigt, wie einfach das auch mit sog. Sparse Volumes funktioniert, die nur so viel Speicher verbrauchen, wie tatsächlich belegt wurde, oder mit denen Volumes realisiert werden können, die größer als der ganze Pool sind.

- ZFS bietet eine gute Grundlage für die effiziente, flexible und einfach zu administrierende Versorgung von OpenSolaris Zonen mit Speicher. Ein Tutorial auf Sun.com zeigt, wie das funktioniert¹⁷.
- Die ZFS-Community hilft: Auf www.opensolaris.org findet sich die OpenSolaris ZFS Community¹⁸. Hier finden sich Dokumentationen, ein Überblick über die Architektur und den Source Code, eine sehr aktive Mailingliste und viele weitere Informationen zu OpenSolaris. Für deutschsprachige Anwender ist auch das Solarium-Blog¹⁹ zu empfehlen.

ZFS: Gestern, heute und morgen

Das OpenSolaris ZFS Dateisystem wurde Mitte 2006 als Teil der Solaris 10 6/06 Distribution für den produktiven Einsatz vorgestellt²⁰. Entwickler haben schon vorher über die Solaris Express bzw. Solaris Express, Developer Edition sowie anderen Solaris-Distributionen Zugriff auf ZFS gehabt, bzw. können heute schon neue Möglichkeiten ausprobieren, die erst in Zukunft für den produktiven Einsatz freigegeben werden.

Zum Jahreswechsel sind mit Solaris 10 11/06 einige Verbesserungen und neue Leistungsmerkmale in das ZFS eingeflossen²¹, etwa rekursive Snapshots, RAID-Z2, Hot-Spares, Clone Promotion, Integration mit dem Solaris Fault Manager, etc. Hin und wieder ändert sich durch ein Update des ZFS Dateisystems auch das Format der Daten in Pools. Diese lassen sich durch `zpool upgrade` schnell und komfortabel in das neue Format überführen.

Zur Zeit wird an weiteren Verbesserungen für ZFS gearbeitet:

- Booten von ZFS erfordert nicht nur die Anpassung des Solaris Boot-Vorganges, sondern auch ein angepasster GRUB-Bootloader (x86/x64 Systeme) bzw. ein angepasstes Open Boot PROM (SPARC) sowie die Anpassung weiterer System-Mechanismen etwa des Installers und der Live Upgrade Funktionalität. Seit OpenSolaris Build 62 ist schon eine Basis-Funktionalität für das Booten von ZFS für x86/x64 Systeme verfügbar, die Interessierte heute schon nutzen können²². Eine volle Boot-Unterstützung wird in einer zukünftigen Version von OpenSolaris erwartet.
- Stand Heute können ZFS Pools beliebig erweitert werden, jedoch lassen sie sich noch nicht in ihrer Kapazität reduzieren. Dies ist zwar eine selten erwünschte Eigenschaft, doch es gibt Szenarien, die diese Funktionalität erforderlich machen (z.B. Irrtümlich an einen Pool angehängte Platten, Umorganisation von Storage in einem SAN, etc.). Heute kann man sich durch Migration von Dateisystemen auf andere, z.B. temporäre Pools mit `zfs send/receive` behelfen, eleganter wäre jedoch ein `zpool remove` Befehl, der Vdevs höherer Ordnung aus einem Pool wieder freigeben kann²³. Dies ist nicht zu Verwechseln mit dem Ersetzen defekter Platten in einem

Mirror oder RAID-Z/Z2 vdev oder dem Freigeben von Spiegelhälften, was heute schon mit `zpool replace` bzw. `zpool remove` möglich ist. Das ZFS Team arbeitet zur Zeit an einer Erweiterung des `zpool remove` Befehls im Sinne der Reduzierung von Pool-Kapazitäten.

- Verschlüsselung von ZFS Dateisystemen: ZFS ist heute schon gegenüber Datenkorruption geschützt und die Checksumme des Überblocks läßt sich mit einer elektronischen Signatur für das gesamte Dateisystem vergleichen. Was noch zu einer wirklich sicheren Speicher-Infrastruktur fehlt, ist die Verschlüsselung von Daten. In Zeiten weit verteilter Infrastrukturen (SAN, iSCSI) und mobiler Geräte (Laptops, USB-Sticks und Festplatten) ist ein wirksamer Schutz vor dem Ausspähen von Daten immer wichtiger. Auch das sichere, schnelle und zuverlässige Löschen von großen Datenmengen ließe sich mit einem verschlüsselten ZFS-Dateisystem elegant lösen: Man löscht einfach den Schlüssel. Die Implementierung einer Verschlüsselungs-Infrastruktur für ZFS ist jedoch aufwändiger als man zunächst denken mag: Während das Verschlüsseln eines Blocks noch als leichte Übung erscheint, stellt sich die Frage: Wie sollen Dateisystem-Schlüssel automatisiert und sicher verwaltet werden? Wie läßt man ein System von einem verschlüsselten Dateisystem booten und woher bekommt es dann sicher einen Schlüssel? Ein OpenSolaris Projekt²⁴ dazu beschäftigt sich gerade mit der Lösung und Implementierung dieser Fragen.
- Seit OpenSolaris Build 62 können ZFS Pools zu Auditing- und Debugging-Zwecken eine Historie von Befehlen speichern²⁵. Ebenfalls seit Build 62 kann neben dem schlanken lzjb-Algorithmus zur Kompression von Blöcken auch der Gzip-Algorithmus in verschiedenen Stärken verwendet werden²⁶.
- Performance-Verbesserungen: Bereits heute ist ZFS aufgrund der Integration von Dateisystem und Volume-Manager, durch seine Copy-on-Write Semantik sowie den intelligenten Prefetch-Algorithmen im ZIO-Modul und durch den aggressiven ARC Cache teilweise um Faktoren von 3-5 schneller als andere Dateisysteme²⁷. Doch es gibt noch Potenzial:
 - Es gibt einige Diskussionen über die NFS-Performance von ZFS. Hier nimmt ZFS den NFS-Commit-Befehl sehr ernst und wandelt ihn um in einen Cache-Flush-Befehl an das Speicher-Subsystem. Dies kann bei grossen Mengen von kleinen Dateien (etwa das Auspacken eines tar-Archivs) dazu führen, dass der Write-Cache von Speicher-Systemen nicht ausgenutzt werden kann. Hier wünschen sich einige Administratoren eine Möglichkeit, die Verantwortung für die Datensicherheit von Write-Caches an das Speicher-System abgeben zu können²⁸.
 - Die ZFS-Performance mit Datenbanken ist im allgemeinen gut, da auch hier die o.a. Mechanismen gut greifen. Speziell für Datenbanken kann man ZFS Block-Größen an die Block-Größe der Datenbank anpassen, um durch diese 1:1-Beziehung Performance zu optimieren. Jedoch haben einige Datenbanken ganz eigene Mechanismen zur Performance-Optimierung, die auf traditionelle Dateisysteme abgestimmt sind. Hier gibt es noch Potenzial, die Performance von ZFS für solche Datenbanken weiter zu optimieren²⁹.

ZFS und Open Source

Als Teil von OpenSolaris ist ZFS unter der OSI-geprüften CDDL-Lizenz³⁰ als Open Source verfügbar und hat eine stetig wachsende und sehr aktive Community hinter sich. Nicht umsonst sind die meisten Fußnoten in diesem Text Verweise auf Community-Beiträge, oft von Nicht-Sun-Mitarbeitern.

ZFS ist seit dem 6. April 2007 auch für das FreeBSD Betriebssystem erhältlich³¹. Auch wird Apple allem Anschein nach ZFS als Teil der nächsten Version von Mac OS X „Leopard“ unterstützen³².

ZFS auf Linux ist noch ein umstrittenes Thema: Es gibt ein Projekt, das ZFS auf FUSE für Linux implementiert³³, doch gegenüber einer Integration von ZFS in den Linux-Kernel gibt es noch rechtliche Bedenken³⁴ seitens der Linux-Community.

Weitere Informationen

Der beste Ort, um Informationen über das OpenSolaris ZFS Dateisystem zu finden ist sicherlich die OpenSolaris ZFS Community:
<http://www.opensolaris.org/os/community/zfs/> .

Dort findet sich unter dem Punkt „Documentation“ oder auch unter docs.sun.com der „ZFS Administration Guide³⁵“.

Das Buch „OpenSolaris für Anwender, Administratoren und Rechenzentren“, erschienen im Springer-Verlag enthält ein Kapitel zu ZFS³⁶.

Das „Solaris Internals“ Buch-Projekt unterhält unter www.solarisinternals.com ein Wiki mit nützlichen Tipps zu ZFS:
http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide .

Schließlich ist auch der Wikipedia-Artikel zu ZFS eine nützliche Informationsquelle: <http://en.wikipedia.org/wiki/Zfs> .

Zusammenfassung

Mit dem Pooled Storage Konzept, Copy-on-Write-Transaktionen, Snapshot- und Clone-Unterstützung, durchgängigen Checksummen, einer modernen, modularen und optimierten Architektur, einfacher Bedienung und vielen weiteren Eigenschaften liefert das OpenSolaris ZFS Dateisystem eine flexible, robuste, mächtige, zuverlässige, modulare und performante Plattform für Dateidienste und öffnet die Tür in die Zukunft der Speicherverwaltung. Als Open Source Dateisystem ist es trotz seines jungen Alters schon fester Bestandteil mehrerer Open Source Betriebssystem-Distributionen und erfreut sich einer stetig wachsenden Entwickler-Gemeinde. Für die Zukunft stehen neben weiteren Performance-Optimierungen auch weiterhin neue und interessante Fähigkeiten im Vordergrund, wie das Booten von ZFS oder die Einbettung von Verschlüsselungs-Technologien, sowie die Adoption durch weitere Betriebssysteme.

- 1 Dave Brillhart: „ZFS: Boils the Ocean, Consumes the Moon“:
http://blogs.sun.com/dcb/entry/zfs_boils_the_ocean_consumes
- 2 Bill Moore, einer der Entwickler von ZFS beschreibt Ditto-Blocks in seinem Blog:
http://blogs.sun.com/bill/entry/ditto_blocks_the_amazing_tape
- 3 Jeff Bonwick, einer der Entwickler von ZFS geht in seinem Blog näher auf diese Eigenschaft ein:
http://blogs.sun.com/bonwick/entry/zfs_end_to_end_data
- 4 Der Autor ist nicht allein: http://blogs.sun.com/chrisg/entry/has_zfs_just_saved_my bzw.
<http://milek.blogspot.com/2006/11/zfs-saved-our-data.html>
- 5 Jeff Bonwick erklärt die Eigenschaften von RAID-Z ausführlich:
http://blogs.sun.com/bonwick/entry/raid_z . Adam Leventhal geht auf die Details von RAID-Z2 hier ein: http://blogs.sun.com/ahl/entry/double_parity_raid_z . Robert Milkowski hat vor kurzem einen Performance-Vergleich von ZFS RAID-Algorithmen gegenüber Hardware-RAID-Controllern veröffentlicht:
<http://milek.blogspot.com/2007/04/hw-raid-vs-zfs-software-raid-part-iii.html>
- 6 ZFS Source Code Tour: <http://www.opensolaris.org/os/community/zfs/source/>
- 7 Wer ZFS im Moment nicht selber ausprobieren will oder kann, findet auf OpenSolaris.org ein paar Demos als Screencast: <http://www.opensolaris.org/os/community/zfs/demos/> . Außerdem gibt es ein unterhaltsames Video, das zeigt, wie man mit ZFS, 3 USB-Hubs und 12 USB-Memory-Sticks eine hochverfügbare Speicherinfrastruktur „für Arme“ machen kann:
http://blogs.sun.com/solarium/entry/solaris_zfs_auf_12_usb .
- 8 Mehr Betrachtungen zur Performance von Mirroring vs. RAID hat Roch Bourbonnais in seinem Blog zusammengefasst: http://blogs.sun.com/roch/entry/when_to_and_not_to
- 9 Richard Elling hat in seinem Blog die Zuverlässigkeits-Eigenschaften von RAID-Varianten für die Sun Fire X4500-Systeme untersucht, die auch für andere Systeme mit vielen Platten gelten: http://blogs.sun.com/relling/entry/raid_recommendations_space_vs_mttdl
- 10 Chris zeigt uns hier, wie er seine eigenen ZFS-Dateisysteme hierarchisch angeordnet hat, um sie besser verwalten zu können:
http://blogs.sun.com/chrisg/entry/where_to_put_zfs_filesystems
- 11 Tims SMF-Snapshot Service: http://blogs.sun.com/timf/entry/zfs_automatic_snapshots_0_8
- 12 Chris' Snapshot-Analysen: http://blogs.sun.com/chrisg/entry/zfs_snapshot_massacre.
- 13 Das nützliche Samba-Snapshot-Skript ist hier zu finden:
http://blogs.sun.com/chrisg/entry/samba_meets_zfs
- 14 Siehe die SMF Community auf OpenSolaris.org:
<http://www.opensolaris.org/os/community/smf/>
- 15 Tims Blog mit seinem SMF ZFS Snapshot Service ist hier:
http://blogs.sun.com/timf/entry/zfs_automatic_snapshots_0_8
- 16 Ben Rockwood beleuchtet in seinem Blog die Vorteile von ZFS und iSCSI:
<http://www.cuddletech.com/blog/pivot/entry.php?id=775>
- 17 „Managing ZFS in Solaris 10 Containers“:
<http://www.sun.com/software/solaris/howtoguides/zfshowto.jsp>
- 18 Die OpenSolaris ZFS Community: <http://www.opensolaris.org/os/community/zfs/>

- 19 Das Solarium-Blog: <http://blogs.sun.com/solarium/>
- 20 Vgl. „What's new in the Solaris 10 6/06 Release“: <http://docs.sun.com/app/docs/doc/817-0547/6mgbdbsmn?a=view>
- 21 Siehe „What's new in the Solaris 10 11/06 Release“: <http://docs.sun.com/app/docs/doc/817-0547/6mgbdbsmb?a=view>
- 22 Eine Anleitung zum Booten von ZFS ist in der OpenSolaris.org ZFS Community erhältlich: <http://www.opensolaris.org/os/community/zfs/boot/>
- 23 Eine lebhaft diskutierte `zpool remove` findet sich in der OpenSolaris Diskussionsliste: <http://www.opensolaris.org/jive/thread.jspa?threadID=22092&tstart=0>
- 24 Das ZFS on Disk Encryption Support Projekt auf OpenSolaris.org: <http://www.opensolaris.org/os/project/zfs-crypto/>
- 25 ZFS On-Disk Format Version 4: <http://www.opensolaris.org/os/community/zfs/version/4/>
- 26 ZFS On-Disk Format Version 5: <http://www.opensolaris.org/os/community/zfs/version/5/>
- 27 Roch Bourbonnais verglich gleich nach Erscheinen das ZFS mit UFS: http://blogs.sun.com/roch/entry/zfs_to_ufs_performance_comparison. Mark Round machte eine unheimliche Begegnung mit dem ARC Cache hier: <http://digitalbadger.net/archives/35-ZFS-and-caching-for-performance.html>. Dennis Clarke sah eine 5-fache Verbesserung der NFS-Performance auf einem Multi-Terabyte-Dateisystem mit Windows/AIX-Klienten: <http://www.opensolaris.org/jive/thread.jspa?threadID=29160&tstart=15>.
- 28 Eine Diskussion zur NFS-Performance findet sich in Rochs Blog: http://blogs.sun.com/roch/entry/nfs_and_zfs_a_fine. Einige Administratoren schalten den ZIL ab, um in solchen Szenarien bessere Performance zu bekommen, wovon Eric Kustarz jedoch abrät: http://blogs.sun.com/erickustarz/entry/zil_disable. Jason Williams empfiehlt als bessere Alternative, im Storage-Subsystem das Ignorieren von Flush-Requests zu konfigurieren: <http://blogs.digitar.com/jjww/?itemid=44>
- 29 Tuning-Tipps für OLTP-Datenbanken mit ZFS und ein paar Performance-Betrachtungen finden sich in Rochs Blog: http://blogs.sun.com/roch/entry/zfs_and_oltp
- 30 Common Development and Distribution License: <http://www.sun.com/cddl/>
- 31 Für mehr Informationen siehe das ZFS FreeBSD Wiki: <http://wiki.freebsd.org/ZFS>
- 32 Kürzlich veröffentlichte „World of Apple“ einen Screenshot des Max OS X Leopard Disk Utility mit der ZFS-Option: <http://news.worldofapple.com/archives/2006/12/17/zfs-file-system-makes-it-to-mac-os-x-leopard/>. Es wird vermutet, daß das zukünftige Time-Machine-Feature von Mac OS „Leopard“ auf ZFS beruht, doch es gibt Stimmen, die das bestreiten: <http://arstechnica.com/staff/fatbits.ars/2006/8/15/4995>
- 33 ZFS auf FUSE/Linux Google Summer of Code Projekt: <http://zfs-on-fuse.blogspot.com/>
- 34 Eine hitzige Debatte zu ZFS auf Linux und die rechtliche Situation dazu ist hier zu finden: <http://www.opensolaris.org/jive/thread.jspa?threadID=28221&tstart=60>
- 35 „The ZFS Administration Guide“: <http://docs.sun.com/app/docs/doc/817-2271>
- 36 Rolf Dietze, Tatjana Heuser, Jörg Schilling: „OpenSolaris für Anwender, Administratoren und Rechenzentren“, erschienen im Springer-Verlag, ISBN 3540292365.