„Best Practices" für Solaris ZFS

Franz Haberhauer
Oracle Deutschland B.V. & Co. KG
Stuttgart

Schlüsselworte:
Solaris, ZFS, Dateisystem

Einleitung

Oracle Solaris bietet mit ZFS ein Dateisystem, das einerseits Volumemanagement integriert, sich vor allem aber durch seine gesicherte Integrität auszeichnet, die durch Transaktionskonzepte und eine konsequente Nutzung von Prüfsummen gewährleistet wird. Integrierte Snapshots, Kompression und ein umfassendes Konzept für Quotas sind Beispiele für die umfangreiche Funktionalität, die zudem ständig erweitert wird Die Möglichkeit ZFS auch für das Root-Filesystem einer Solaris-Installation zu nutzen eröffnet elegante Möglichkeiten insbesondere für Betriebssystem-Upgrades. Ebenso elegant kann durch Nutzung von Flash-Memory und SSDs in sogenannten Hybriden Storage Pools die Performance effizient und transparent für Anwendungen optimiert werden.

Bei der Entwicklung von ZFS war ein Ziel, die Notwendigkeit von Tuning-Maßnahmen zu minimieren. Konfigurationsentscheidungen haben aber sehr wohl Einfluss auf die Performance und einige Lasten - insbesondere DBMS - profitieren von geeigneten Einstellungen. Einstellungen und Algorithmen im ZFS werden gelegentlich an Erfahrungen aus dem praktischen Betrieb angepasst und man sollte deswegen ältere Tuning-Hinweise immer wieder hinterfragen.

Im Folgenden wird zunächst ein Überblick über ZFS gegeben und der aktuellen Stand der Entwicklung beleuchtet. Danach folgen Hinweise zur Konfiguration und es werden "Best Practices" und aktuell immer wieder gestellte Fragen diskutiert.

ZFS Überblick

Mit ZFS führte Sun vor mittlerweile fast fünf Jahren ein neues lokales Dateisystem ein, das die klassischen Aufgaben eines Dateisystems - Daten ohne großen administrative Aufwand schnell und effizient zu schreiben, sicher zu speichern und wieder auszulesen - mit einer neuen innovativen Architektur grundsätzlich anders löst als traditionelle Dateisysteme. Dateisystem und Volumemanagement sind integriert, wodurch sich nicht zuletzt die starken Mechanismen zur Sicherung der Datenintegrität realisieren lassen, die ZFS auszeichnen. Dazu gehören die konsequente Nutzung des Transaktionskonzepts und von Prüfsummen. Integrierte Snapshots, Kompression, ein umfassendes Konzept für Quotas sind Beispiele für die umfangreiche Funktionalität, die zudem ständig erweitert wird - so wurde Ende 2009 in OpenSolaris Deduplication eingeführt. Die Möglichkeit ZFS auch für das Root-Filesystem zu nutzen, eröffnet elegante Möglichkeiten insbesondere für Betriebssystem-Upgrades. Das neue Image Packaging System zur Paketverwaltung in OpenSolaris und künftig Solaris 11 nutzt diese Mechanismen und daher ist ZFS in diesen Versionen als Root-Filesystem vorgegeben. Eine einfache Möglichkeit von den Vorteilen von ZFS zu profitieren sind die Speichersysteme der Oracle Sun Storage 7000 Unified Storage Systems Produktlinie, die als Appliances auf OpenSolaris basieren und damit neueste ZFS Funktionalitäten mitbringen.

Warum war es überhaupt an der Zeit ein neues lokales Dateisystem zu entwicklen? Viele der heute eingesetzten Dateisysteme wurden vor 20 bis 40 Jahren konzipiert und selbst neuere Dateisysteme folgten meist den etablierten Konzepten. Durch innovative Ansätze können Zuverlässigkeit,

Performance und Skalierbarkeit aber deutlich gesteigert und die Administration wesentlich vereinfacht werden.

Ein zentrales historisches Architekturkonzept ist die 1:1-Abbildung zwischen einem Dateisystem und einem logischen Volume, das von der physischen Struktur des zugrunde liegenden Speichersystems abstrahiert. Über eine eigene Software-Schicht, einen Volume-Manager, können mehrere Platten zu einem großen virtuellen Laufwerk zusammenfügt werden. Im Hinblick auf Performance kann dabei Striping oder im Hinblick auf eine höhere Verfügbarkeit Spiegelung oder ein höheres RAID-Level eingesetzt werden. Diese Abstraktion vereinfacht zunächst die Implementierung eines Dateisystems, verhindert aber andererseits Optimierungen, die ein durchgängiges System unter Kenntnis der physischen Struktur automatisch umsetzen könnte. Oft gibt es dazu manuell zu konfigurierende Tuning-Optionen und -Parameter. Zudem sind Dateisystem und Volume-Manager durch die explizite Schnittstelle dazwischen aufwändig separat und explizit zu verwalten.

In den letzten Jahrzehnten hat sich auch die Hardware-Umgebung signifikant geändert. In vielen Dateisystemen spiegelt sich immer noch das Modell einer Platte mit einer festen Anzahl von Sektoren je Zylinder wieder, was durch Zone Bit Recording seit langem überholt ist: die äußeren Zylindern heutiger Festplatten enthalten mehr Sektoren, als die inneren. Auch die Leistungscharakteristika haben sich geändert: Vor 15 Jahren hatte eine typische Platte eine Kapazität von 1GB und bei 5400 Umdrehungen pro Minute eine mittlere Zugriffszeit von 11 Sekunden, womit sie bei wahlfreiem Zugriff 90 IOPS abarbeiten konnte, eine schnelle aktuelle Platte hat 300GB Kapazität, 15.000rpm, 5ms mittlere Zugriffszeit und schafft 200 IOPS. Die Transferrate, die bei sequentiellen Zugriffsmustern zum Tragen kommt, wuchs von 3 auf 90MB/s. Während Kapazität und der Durchsatz bei sequentiellen Zugriffsmustern um Größenordnungen angewachsen sind, hat sich die Bandbreite für wahlfreien Zugriff gerade verdoppelt. Die CPU-Leistung hat sich in dieser Zeit ebenfalls mehr als verhundertfacht, so dass heute mehr CPU-Instruktionen im Kontext von E/A-Operationen genutzt werden können ohne die I/O-Performance signifikant zu beeinträchtigen.

Ein weiteres Defizit traditioneller Dateisysteme liegt darin, daß die Datenintegrität nicht durchgängig gewährleistet wird. Strukturelle Operationen wie das Anlegen oder Löschen einer Datei erfolgen nicht in einer einzigen, sondern in einer Abfolge von Schreiboperationen – Blöcke werden zum Beispiel in Freilisten aus- oder eingetragen, die Existenz von Dateien in Verzeichnissen vermerkt. Um potentiell bei einem Systemabbruch (Software-Panic oder Stromausfall) entstehende Inkonsistenzen zu bereinigen, wurde zunächst über ein Hilfsprogramm (fsck) beim Systemstart die strukturelle Konsistenz geprüft und gegebenenfalls wieder hergestellt. Mit zunehmend größeren Dateisystemen dauerte dies aber immer länger. Daher wurde Logging eingeführt, bei dem strukturelle Änderungen als Transaktionen in einem Log protokolliert werden, der dann beim nächsten Systemstart abgearbeitet wird. Dennoch gibt es insbesondere bei Hardware-Fehlern Szenarien, in denen dieser Log verworfen werden muss und ein fsck-Lauf nötig wird, der sich dann über Stunden oder gar Tage hinziehen kann. Auch die Trennung zwischen Dateisystem und Volume-Management bringt nicht behandelbare Fehlerszenarien mit sich. Werden z.B. aufgrund von Hardware-Problemen korrupte Daten gelesen, führen diese entweder, falls es sich um Strukturinformationen handelt, oft zu einem Systemabbruch im Dateisystemcode oder bei Anwendungsdaten zu Problemen in der Anwendung. Das Dateisystem kann solche korrupten Daten nicht erkennen und selbst falls doch, kann es auch wenn ein gespiegeltes Volume vorliegt, den Volume Manager nicht instruieren, möglicherweise korrekte Daten von der anderen Spiegelhälfte zu lesen. Stille Datenkorruption durch transiente Hardware-Defekte oder Software-Fehler, die erst spät aufgrund gehäufter Anwendungsprobleme und Systemabstürzen auffällt, ist ein reales Risiko, das bedingt durch die heutigen Datenvolumina kein unwahrscheinliches Ereignis mehr ist [1,2,3].

ZFS Architektur

Mit dem Solaris ZFS wurde ein neuartiges Dateisystem entwickelt mit Schwerpunkten auf einfache Verwaltung, durchgängige Sicherstellung der Datenintegrität und immense Skalierbarkeit.

Beim ZFS wurde die 1:1-Abbildung zwischen Dateisystemen und Volumes verworfen. Stattdessen werden aus physischen oder virtuellen Laufwerken (LUNs) sogenannte Storagepools (ZPools) angelegt, in denen jeweils mehrere Dateisysteme angelegt werden können. Ein solches Dateisystem besteht lediglich aus einem Mount-Punkt und einer Reihe von Attributen (Properties), z.B. Reservierung, Quota, Kompression etc., nicht aber aus aus einem zu initialisierende festen Speicherbereich. Im Storagepool wird die Kapazität für alle Dateisysteme gemeinsam verwaltet. Dadurch gibt es keine Fragmentierung der verfügbaren Speicherkapazität in verschiedene Dateisysteme: das aufwändige Verkleinern und Vergrößern von Dateisystemen entfällt. Ein Dateisystem hat keine vorbestimmte Größe, seine aktuelle Größe wird durch die Menge der gespeicherten Daten bestimmt. Die freie Kapazität ergibt sich aus der freien Kapazität des Storagepool. Optional kann sie durch eine Quote beschränkt werden. Durch Reservierungen kann zudem für ein Dateisystem eine bestimmte Mindestkapazität garantiert werden, indem für dieses Dateisystem die reservierte Kapazität im Storagepool frei gehalten wird. Ein Dateisystem wird beim Anlegen automatisch über einen Standardpfad in den Dateisystembaum eingebunden – es ist aber auch möglich, Mount-Punkte explizit zu spezifizieren. ZFS-Dateisysteme können hierarchisch geschachtelt werden.

Storagepools (ZPools) bestehen aus virtuellen Geräten (vdevs), die Raw-Devices oder Spiegel (RAID-1) bzw. RAID-Z-Gruppen aus mehreren Raw-Devices sein können. Insbesondere für Testzwecke können statt Raw-Devices auch Dateien in Dateisystemen verwendet werden. RAID-Z ist eine dynamische Variante von RAID-5: RAID-Z nutzt variable Stripe-Breiten um alle Schreiboperationen als Full-Stripe-Writes ausführen zu können und damit das sonst nötige Einlesen vorhandener Daten zur Berechnung der Parity-Information zu vermeiden. Die Blockgröße ist variabel zwischen 512 Byte und 128KB – wobei die Voreinstellung bei 128KB liegt. Damit wird zudem das sogenannte RAID-5-Write-Hole - mögliche Datenkorruption durch partielles Ausschreiben von Daten und Parity etwa bei einem Stromausfall - vermieden. RAID-Z ermöglicht die sichere und performante Implementierung eines höheren RAID-Level ohne nichtflüchtigen Speicher (NVRAM), was die Implementierung sehr preisgünstiger Speicherlösungen auf der Basis von JBODs erlaubt. Warum wurde ein derart einfaches und vorteilhaftes Konzept nicht schon früher implementiert? Entscheidend ist die Rekonstruktion von Daten: Es gibt keine einfache Formel wie „XOR der Bits aller Platten auf 0“. Dateisystem-Metadaten werden traversiert, um die RAID-Z Geometrie zu ermitteln. Ein solcher Ansatz ist bei einer Trennung der logischen Dateisystem-Ebene und der physischen Ebene der Volumes nicht möglich. Ein weiterer Vorteil des integrierten Ansatzes von ZFS liegt darin, dass eine Initialisierung oder ein „Resilvering“ von Spiegeln sehr schnell ist, falls das Dateisystem nicht ganz gefüllt ist: es werden nämlich nur echte Nutzdaten auf neue Spiegel kopiert, es muss nicht die gesamte Kapazität eines Spiegels initialisiert oder kopiert werden. Sind aktuelle Platten allerdings voll, dann kann ein “Resilvering” Stunden oder gar Tage dauern [7]. Um auch während der Rekonstruktion eines Laufwerks Schutz gegen weitere Ausfälle zu bieten, wurden zunächst RAID-Z2 und dann RAID-Z3 (mit Solaris 10 9/10) eingeführt, die durch eine doppelte bzw. dreifache Parity den Ausfall von zwei bzw. drei Laufwerken verkraften [4].

Storagepools können von einem Rechner exportiert und auf einem anderen wieder importiert werden, da die Strukturinformationen in den Pools selbst gespeichert sind. Im Unterschied zum UFS ist ZFS dabei unabhängig von der Endianess des Rechners. Daten werden jeweils mit der nativen Endianess geschrieben. Bei Bedarf werden beim Lesezugriff Bytes entsprechend getauscht.

ZFS bietet eine POSIX-konforme Dateischnittstelle. Für Access Control Lists (ACLs) wurde die ACL-Spezifikation von NFS Version 4 implementiert, die unter anderem feiner granulierte Privilegien

und eine umfassendere Semantik für die Vererbung von Verzeichnissen auf Unterverzeichnisse bietet als der im Solaris UFS implementierte POSIX-Entwurf. Diese umfassendere Spezifikation erlaubt zudem eine bessere Interoperabilität mit der Windows-Welt.

Neben der Dateischnittstelle bietet ZFS eine emulierte Geräteschnittstelle über die sogenannte zvols, die als Raw-Devices unter /dev z.B. als Swap-Bereich oder für DBMS bereitgestellt werden wobei ein Storagepool die Speicherkapazität liefert.

In einem ZFS können bis zu 2^{128} Byte Speicherkapazität verwaltet werden, womit die Kapazität für praktische Zwecke unbegrenzt ist. Bei solchen Kapazitäten muss die strukturelle Konsistenz des Dateisystems auf der Platte immer sichergestellt sein. Das wird erreicht, indem alle Operationen als Copy-on-Write-Transaktionen aufgeführt werden. Geänderte Daten werden niemals überschrieben sondern immer neu ausgeschrieben. Dabei werden variable Blockgrößen von bis zu 128KB verwendet, was eine automatische Optimierung ermöglicht. ZFS verfügt über eine I/O-Pipeline mit einem Scheduler, der unter Berücksichtigung logischer Abhängigkeiten über Prioritäten und Deadlines die Abfolge von E/A-Operationen optimiert: Operationen können parallelisiert, umgeordnet oder sortiert werden. So werden Leseoperationen normalen asynchronen Schreiboperationen vorgezogen, die innerhalb der letzten halben Sekunde abgesetzt wurden, um auch bei hoher Last ein reaktives Systemverhalten zu erreichen . ZFS erkennt sequentielle oder andere regelmäßige (z.B. jeder n-te Block) Zugriffsmuster und liest entsprechend Daten voraus. Für Anwendungen, die konzeptionell mit festen Blockgrößen arbeiten - wie etwa DBMS – kann für ein Dateisystemen eine feste Blockgröße vorgegeben werden. Dateien sind in einer Baumstruktur organisiert, wobei die Daten in den Blättern liegen und die Knoten Verweise mit Prüfsummen der verwiesenen Objekte enthalten. Der Algorithmus zur Berechnung der Prüfsummen ist konfigurierbar, verfügbar sind u.a. fletcher4 (voreingestellt) oder SHA256. Dadurch, dass die Prüfsummen nicht nur lokal für einzelne Blöcke gelten, können nicht nur gekippte Bits identifiziert werden, sondern auch komplexere Fehler, wie verlorengegangene oder fehlgeleitete Schreiboperationen, die aus Treiber-, Hardware- oder auch administrativen Fehlern resultieren können. Werden korrupte Daten entdeckt können bei einer redundanten Speicherung in einem Spiegel oder RAID-Z korrekte Daten rekonstruiert werden, falls die Daten nur auf einem Laufwerk korrumpiert wurden. Die korrigierten Daten werden dann unmittelbar zurückgeschrieben, um so Software- oder transiente Fehler zu adressieren. Erst, wenn das nicht erfolgreich möglich ist, wird ein Laufwerk als solches als defekt markiert. Um Datenkorruption möglichst frühzeitig zu entdecken und zu beheben, besteht die Möglichkeit durch einen im Hintergrund laufenden Prozess, den sogenannten Scrubber, periodisch die Prüfsummen und damit die Konsistenz der Daten validieren zu lassen. Damit lässt sich erstmals das Risiko stiller Datenkorruption effektiv und effizient adressieren. Blöcke können nicht nur auf der physischen Ebene durch Spiegel oder RAID-Z redundant gespeichert werden, sondern werden zudem logisch bis zu dreimal in sogenannten Ditto-Blöcken repliziert. Blöcke werden durch Disk Virtual Addresses (DVA) identifiziert. Ein Verweis auf einen Block kann drei DVA aufnehmen. Benutzerdaten werden einfach, Dateisystem-Metadaten doppelt und Daten, die für alle Dateisysteme in einem Storagepool relevant sind dreifach gespeichert, wobei die physische Lage unter Verfügbarkeits- und Performance-Gesichtspunkten optimiert wird. Der zusätzliche Platzbedarf liegt unter 2%.

Änderungen werden von den Blättern zum Wurzelknoten, dem sogenannten Uberblock, propagiert. Das Ausschreiben des Uberblock ist der atomare Zustandsübergang, der das Ende einer Transaktion definiert. Aus Performance-Gründen werden Änderungen zu Transaktionsgruppen zusammengefasst. Periodisch – voreingestellt sind fünf Sekunden – wird eine solche Transaktionsgruppe abgeschlossen und alle geänderten Blöcke werden eine Ebene nach der anderen bis zum Uberblock ausgeschrieben. Währenddessen laufen weitere Änderungen in eine neue Transaktionsgruppe. Erst wenn diese geschlossen wird, die vorige ihre Synchronisation auf Platte aber noch nicht abgeschlossen haben sollte, werden weitere Schreiboperationen von Anwendungen blockiert, um die Last auf dem E/A-System zu reduzieren. Um synchrone Operationen wie sie etwa in DBMS genutzt werden schnell

abwickeln zu können, wird zudem ein Intent Log genutzt. Durch den Copy-on-Write-Mechanismus werden wahlfreie Schreibzugriffe in effiziente große, sequentielle Schreiboperationen umgewandelt, was den Zugriffscharakteristika moderner Platten entgegen kommt.

Wird nun der ersetzte Wurzelknoten nicht freigegeben, sondern sichtbar gemacht, erhält man eine Snapshot-Funktionalität. ZFS ermöglicht so eine unbegrenzte Zahl von Snapshots, die den Zustand eines Dateisystems zu einem bestimmten Zeitpunkt als nicht änderbare Kopie darstellen. Snapshots können verwendet werden, um einen konsistenten Zustand eines Dateisystems zu sichern. Für eine besonders effiziente Datensicherung kann ein Hilfsprogramm, die Unterschiede zwischen zwei Snapshots serialisieren (die Baumstruktur des Dateisystems in einen Datenstrom verwandeln, der in eine Datei oder auf Band geschrieben oder über ein Netzwerk transferiert werden kann). Damit kann dann z.B. ein Dateisystem auf einem entfernten Rechner auf logischer Ebene zeitnah repliziert werden. Der aktuelle Zustand eines Dateisystems kann auf den Stand eines Snapshot zurückgesetzt werden.

Da ersetzte Uberblocks nicht sofort verworfen werden, können sie dazu verwendet werden, ein Dateisystem, das durch ein Fehlverhalten von Hardware unmittelbar vor einem Systemcrash korrumpiert wurde, auf einen einige Sekunden älteren aber konsistenten Stand zurückzusetzen. Ein typisches Szenario sind Virtualisierungslösungen, die E/A-Reihenfolgen nicht einhalten (Cache Flush Kommandos ignorieren) und etwa einen neuen Uberblock auf stabilen Speicher schreiben noch bevor alle zugehörigen Daten ausgeschrieben sind. Kommt es dann zu einem Crash ist der ZPool inkonsistent, was beim Import erkannt wird, so dass der Pool nicht geöffnet wird. Ab Solaris 10 9/10 versucht ZFS eine automatisierte Recovery eines solchen Pools.

Auf der Basis eines Snapshot kann ein sogenannter Clone eines Dateisystems eingerichtet werden, der auch geändert werden kann. Durch solche Clones können Varianten von Dateisystemen, die sich nur in Teilbereichen unterscheiden, etwa Arbeitsbereiche in der Software-Entwicklung oder Installationsvarianten sehr effizient gespeichert werden. Wird ZFS als Root-Dateisystem [5] genutzt, kann diese Clone-Funktionalität genutzt werden, um speichereffizient und schnell mehrere Boot-Umgebungen anzulegen und zu verwalten. In OpenSolaris und damit auch Solaris 11, wo ZFS als Root-Dateisystem gesetzt ist, werden diese Mechanismen in Verbindung mit dem neuen Image Packaging System (IPS) genutzt, um auf sichere Art und Weise Änderungen in das System einzubringen, die dann im Falle eines Falles einfach verworfen werden können.

ZFS erlaubt es, Daten optional komprimiert abzuspeichern. Die Kompression erfolgt auf Blockebene. Als Algorithmus wird LZJB verwendet, ein einfaches, schnelles Verfahren, dass sich nur unwesentlich auf die Schreib-/Lese-Performance auswirkt und doch Kompressionsfaktoren zwischen 1,5 und 4 erzielt. Alternativ sind verschiedene GZIP-Varianten verfügbar, die typischerweise eine höhere Kompression erreichen, dafür aber die E/A-Performance messbar reduzieren [6]. Andererseits werden durch die Reduktion des E/A-Volumens manche Lasten sogar beschleunigt.

Solaris 11 Express sowie die aktuelle „Firmware“ für die Speichersysteme der S7000-Reihe bieten zudem die Möglichkeit Deduplication zu nutzen [7]. Da ZFS ohnehin für jeden Block eine Prüfsumme errechnet, liegt es nahe, diese auch als Grundlage für Deduplication zu nutzen. Blöcke mit derselben Prüfsumme werden als identisch angesehen und nur einmal physisch gespeichert. Bei einem sicheren Hash-Verfahren wie SHA256 ist eine fälschlichen Gleichsetzung um 50 Größenordnungen weniger wahrscheinlich als ein von ECC unentdeckter Hauptspeicherfehler. Bei einfacheren Verfahren wie fletcher4 empfiehlt sich zusätzlich ein einmaliger Vergleich der Blockinhalte. Nach einer Aktivierung werden neu angelegt Blöcke der Deduplizierung unterworfen. Dazu wird eine Tabelle im Hauptspeicher aufgebaut, die je Eintrag etwa 300 Byte umfasst. Bei einem voll deduplizierten Pool mit 10TB sind das bei einer Blockgröße von 128KB 0,23% Overhead - andererseits sind das 23GB RAM. Bei eine Blockgröße von 8KB sind es 3,75% bzw. 375GB!

Eine weitere Innovation in ZFS ist die Integration schnellen Flash-Speichers insbesondere in Form von Solid State Disks (SSDs) in die Architektur - die Hybrid Storage Pools [8]. Hierbei werden SSDs

genutzt, einerseits um die Performance-kritischen synchronen Schreiboperationen zu beschleunigen indem der ZFS Intent Log (ZIL), auf solche schnellen Medien gelegt wird, idealerweise Typen, die besonders für Schreiboperationen optimiert sind,. Andererseits wurde um Leseoperationen zu beschleunigen ein optionaler Second Level Filesystem Cache eingeführt, der L2ARC, der ebenfalls auf SSDs gelegt werden kann. Dadurch können für Lasten, die ein begrenztes Working Set verwenden, für preisgünstige, große aber vergleichsweise langsame Platten verwendet und dennoch mittels eines L2ARC eine überlegene Lese-Performance erreicht werden, wobei für den L2ARC eine für das Working Set ausreichende Anzahl von SSDs genutzt wird.

ZFS Best Practices

Konfiguration

Ein ZPool aus mehreren Platten kann in unterschiedlichen RAID-Konfigurationen aufgebaut werden, die sich im Hinblick auf Redundanz und Performance unterscheiden. Dies ist insbesondere relevant bei RAIDZ-Konfigurationen. Bei RAIDZ ist zu entscheiden wieviele Parity-Platten verwendet werden sollen. Aufgrund der Entwicklung der Platten-Kapazitäten dauert die Integration einer Ersatzplatte in einen RAID-Verbund heute zumindest mehrere Stunden [4]. Um zu vermeiden, dass es innerhalb dieser Zeit durch einen weiteren Plattenfehler zu Datenverlust kommt, sollte zumindest ein RAIDZ2 mit Double-Parity verwendet werden. Eine weitere Entscheidung bei RAIDZ ist, ob man breite vdevs oder besser eine Konkatenation mehrerer kleinerer vdevs konfiguriert. Innerhalb eines RAIDZ-vdev wird ein ZFS-Block über möglichst viele Laufwerke verteilt geschrieben (minimal 512 Byte/Platte). Damit triggert das Schreiben oder Lesen eines Blocks eine E/A-Operation auf jeder Platte, d.h. die maximale Rate an Random-Zugriffen (gemessen in IOPS) entspricht der einer einzelnen Platte [11]. Insofern ist für ZPools mit wahlfreien Zugriffen eine Konkatenation kleinerer RAIDZ2-vdevs mit jeweils 5-9 Datenplatten, 2 Parity-Platten je vdev und einigen Hot-Spare-Platten für den Pool zu empfehlen. Bei 46 1TB-Platten mit je ca. 200IOPS wären dann mögliche Konfigurationen:

- 5x(7+2), 1 hot spare, 17.5 TB, 1.000 IOPS
- 4x(9+2), 2 hot spares, 18.0 TB, 800 IOPS
- 6x(5+2), 4 hot spares, 15.0 TB, 1.200 IOPS

Bei konkatenierten vdevs sorgt ZFS für einen Lastverteilung (dynamisches Striping) über die vdevs.

Bei der Dimensionierung von ZPools sollte beachtet werden, dass die Performance von “Luft zum Atmen” profitiert. ZPools, in denen Dateien ständig gelöscht und neu angelegt werden, sollten maximal zu 80% gefüllt werden, ansonsten kann es aufgrund anderer Allokierungsalgorithmen, die dann greifen, zu einer reduzierten Performance kommen. Dateisysteme, in die lediglich geschrieben, Dateien aber nicht gelöscht werden, sind davon nicht betroffen. Der Schwellwert ist konfigurierbar (`metaslab_df_free_pct`[10]).

ZFS Intent Log (ZIL)

Die Latenz synchroner Schreiboperationen kann signifikant verbessert werden, indem der ZFS Intent Log (ZIL) auf ein schnelles stabiles Speichermedium gelegt wird. Hierfür bietet sich Flash-Memory insbesondere in Form von SSDs an. Der ZFS Intent Log muss maximal des E/A-Volumen zweier Transaktionsgruppen, d.h. von ca. 10 Sekunden aufnehmen, da im ZIL Änderungen durch synchrone Schreiboperationen protokolliert werden, die dann beim Ausschreiben der Transaktionsgruppen in den Datenbereichen nachgezogen werden. Daher wird der. ZIL im Normalbetrieb auch lediglich geschrieben und nur nach einem Crash gelesen. Die minimale Größe ist 64MB. Um zu verhindern,

dass durch den Ausfall eines Log Device vermeintlich auf stabilen Speicher geschriebene Informationen verloren gehen, sollte der ZIL immer auf gespiegelten Laufwerken angelegt werden.

Liegt der Storagepool insgesamt auf einem NVRAM-gepufferten Speicher – z.B. großen Speichersystemen, ist ein separierter ZIL nicht nötig – ein ZIL ist jedoch immer vorhanden. Bei hohen Raten synchroner Schreiboperationen, die von der Latenz her nicht kritisch sind, - wie etwas bei Oracle-Tablespaces - kann es dann sinnvoll sein, das doppelte Ausschreiben der Daten einmal in den Log und zum zweiten in die Datenbereiche zu vermeiden und stattdessen die Daten gleich in die Datenbereiche zu schreiben ohne jedoch den Uberblock zu modifizieren und im ZIL lediglich die Referenzen zu vermerken. Hierdurch kann ein Hotspot durch den ZIL vermieden und die insgesamt benötigte E/A-Bandbreite fast halbiert werden. Ab Solaris 10 9/10 und in der S7000-Reihe gibt es zur Steuerung auf Dataset-Ebene das Property `logbias` [10]. In älteren ZFS-Versionen kann das Verhalten über den globalen Parameter `zfs_immediate_write_sz` gesteuert werden [11].

Von einer Deaktivierung des ZIL (`zil_disable`), die in Foren insbesondere für NFS-Server gelegentlich empfohlen wird, ist dringend abzuraten, da hierdurch die Datenintegrität bei einem Crash gefährdet wird [11].

ZFS auf Speichersystemen

ZFS eignet sich besonders, um aus JBODs insbesondere in Verbindung mit SSDs leistungsfähige Speichersysteme aufzubauen. Natürlich lassen sich die administrativen Vorteile auch in Verbindung mit leistungsfähigen Speichersystemen, die Hardware-RAIDs und batteriegepufferte Caches enthalten, nutzen. Allerdings kommen hier die Stärken von ZFS im Hinblick auf eine End-to-End-Datenintegrität nur zum Tragen, wenn auf der ZFS-Ebene eine redundante Datenhaltung konfiguriert ist, so dass ZFS etwa bei Fehlern in HBAs Daten automatisch korrigieren kann, was bei einer Redundanz nur über ein RAID im Speichersystem nicht möglich ist. Auch das automatische Tuning von ZFS greift nur eingeschränkt, da hier bei einer LUN die typische Leistung eines Plattenlaufwerks unterstellt wird, während eine LUN, die von einem Speichersystem exportiert wird, wesentlich leistungsfähiger sein kann (ab 10 Laufwerken hinter eine LUN Konfiguration via `zfs_vdev_max_pending` [10,11]). Seit Solaris 10 5/09 versucht ZFS Speichersysteme mit NVRAM-Caches zu erkennen und ggf. an diese keine Cache Flush Instruktionen zu schicken. Generell sollten solche Instruktionen von Speichersystemen mit NVRAM ignoriert werden, was allerdings nicht immer der Fall oder manchmal auch konfigurierbar ist. Falls Solaris solche Speichersysteme nicht automatisch erkennt, kann diese Information konfiguriert werden [11]. Das gilt auch für Flash-Speicher und SSDs, die intern über Kondensator-gestützte DRAM-Caches verfügen [12].

ZFS Caches: ARC und L2ARC

Der ZFS Hauptspeicherpuffer, der nach dem Verwaltungsalgorithmus als ARC – Adaptive Replacement Cache – bezeichnet wird, ist in seiner Größe variabel und wird von ZFS automatisch den Erfordernissen angepasst. Werden auf einem System Lasten gefahren, die kurzfristig große Hauptspeicherbereiche anfordern wie Datenbanken beim Start oder speicherintensive Anwendungen, ist es empfehlenswert, die maximale Größe des ARC zu beschränken (`zfs_arc_max`[11]).

Mit dem L2ARC besteht optional die Möglichkeit auf schnellem Speicher eine zweite Ebene für den Dateisystempuffers einzurichten. Dabei ist zu beachten, dass zu dessen Verwaltung Hauptspeicher benötigt wird – ca. 200 Byte je Block im L2ARC. Während dies bei der voreingestellten Blockgröße von 128KB nicht ins Gewicht fällt, ist der Speicherbedarf bei einer Blockgröße von 8KB dagegen durchaus signifikant: bei einem L2ARC von 150GB sind das 3.75GB im ARC (2,5%). Bei der Performance-Analyse eines L2ARC ist zu beachten, dass es typischerweise etliche Stunden dauert, bis sich dieser in einem eingeschwungenen Zustand befindet.

Über die Properties `primarycache` und `secondarycache` kann feingranuliert je Dataset gesteuert werden, ob Daten, nur Metadaten oder gar nichts im ARC bzw. L2ARC gehalten werden soll. Darüber kann eine Doppelpufferung von Daten etwa bei Datenbanken oder anderen Anwendungen, die eine eigene Datenverwaltung haben, wie etwa Out-of-Core-Solver in HPC-Anwendungen, vermieden werden.

Unter dem Aspekt der Vermeidung von Doppelpufferung ist das Property `primarycache` das Äquivalent zum UFS Direct I/O. Weitere Funktionalitäten von UFS Direct I/O wie die Aufhebung der Serialisierung von Schreiboperationen vor dem Single Writer Lock sind bei ZFS unabhängig von der Pufferverwaltung adressiert.

ZFS und DBMS

Bei DBMS ist zu unterscheiden, ob diese ein optimierendes Speicherkonzept haben wie Oracle, Informix oder DB/2 oder auf eine Datenhaltung in Dateien abzielen wie Sybase, MySQ oder PostGreSQL.

Bei der ersten Gruppe arbeiten manchmal E/A-Optimierungen in ZFS und die im DBMS gegeneinander, während die zweite Gruppe typischerweise von den E/A-Optimierungen im Dateisystem voll profitiert. Bei Beachtung einiger Grundregeln kann auch für die erste Gruppe von DBMS auf ZFS eine gute Performance erzielt werden [10,13]. Trigger für den Einsatz von ZFS sind hier insbesondere die mächtigen administrativen Funktionen von ZFS und deren einfache Nutzung weniger maximale Performance.

Für Daten und Logs sollten getrennte ZPools angelegt werden, für Logs als Blockgröße die voreingestellten 128KB verwendet werden, für die Daten dort, wo in den DBMS Random-Zugriffe dominieren die Blockgröße der des DBMS angepasst werden. Werden Daten überwiegend sequentiell bearbeitet (Full Table Scans, Direct Path Sorts) kann allerdings die Voreinstellung von 128KB wiederum vorteilhaft sein. Durch eine reduzierte Blockgröße werden einige der Optimierungen, für die DBMS alternative Lösungen haben, im ZFS deaktiviert [10,13]. Einige DB-Lasten profitieren von Kompression. Oracle 11gR2 hat jetzt ebenfalls die Möglichkeit zu komprimieren und als Oracle Flash Cache Flash-Speicher oder SSDs als schnelle Speichermedien einzusetzen. Wenn solche Funktionen im DBMS verfügbar sind, kann das DBMS diese typischerweise zielgerichteter und damit effizienter einsetzen als ZFS. In diesem Sinne empfiehlt sich auch die Vermeidung von Doppelpufferung über das ZFS Property `primarycache=metadata`. Da insbesondere Oracle alle E/A-Operationen synchron schreibt, profitiert Oracle von einem separierten ZIL auf einem schnellen Speicher und/oder von NVRAM-gepufferten Speichersystemen (ZFS-Property `logbias=throughput`). Abschliessend noch der Hinweis darauf, dass ZFS ein lokales Dateisystem ist und damit nicht als Shared-Filesystem für Oracle RAC geeignet ist.

Zusammenfassung

ZFS ist ein modernes Dateisystem, das über vielfältige Funktionalitäten verfügt, die zudem ständig erweitert werden. ZFS kann neue Hardware-Technologien wie schnellen, stabilen Flash-Speicher effektiv und transparent nutzen. Da manuelles Tuning weitgehend vermieden werden soll, werden auch interne Algorithmen ständig verfeinert und Parameter justiert. Neueste Funktionalitäten und Optimierungen finden sich zunächst in OpenSolaris/Solaris 11 Express und bedingt durch den längerfristigen Release-Zyklus mit einer mehr oder weniger langen zeitlichen Verzögerung in Solaris 10 (Updates). Ein alternativer Weg neueste ZFS-Funktionalitäten zügig produktiv nutzen zu können, sind die OpenSolaris-basierten Speichersysteme der Oracle Sun Storage 7000 Unified Storage Systems Produktlinie. Insofern ist bei neueren Funktionalitäten, aber auch bei Tuning-Empfehlungen und anderen „Best Practices“ darauf zu achten, ob diese allgemeingültig oder erst ab einem bestimmten Solaris Update, OpenSolaris Build oder

S7000-Firmware-Stand[14] verfügbar sind. Neben der ZFS-Dokumentation zu Solaris 10 bzw. Solaris 11 Express auf docs.sun.com sind insbesondere die ZFS Guides auf solarisinternals.com („Best Practices Guide“, „Evil Tuning Guide“, „Troubleshooting Guide“) als aktuelle Informationsquellen empfehlenswert. Die Konfiguration von Oracle DBMS auf ZFS wird in einem Oracle White Paper [10] näher beleuchtet.

Literaturverweise

1. Robin Harris: *CERN's Data Corruption Research*, 17.09.2007, http://storagemojo.com/2007/09/19/cerns-data-corruption-researc

2. Bernd Panzer-Steindel: *Data Integrity*, CERN/IT, 08.04.2007, http://indico.cern.ch/getFile.py/access?contribId=3&sessionId=0&resId=1&materialId=paper&confId=13797

3. Yupu Zhang, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau: *End-to-end Data Integrity for File Systems: A ZFS Case Study*, 8th USENIX Conference on File and Storage Technologies FAST10, 2010, http://www.usenix.org/events/fast10/tech/full_papers/zhang.pdf

4. Adam Leventhal: The Need for Triple Parity RAID, 21.12.2009, http://dtrace.org/blogs/ahl/2009/12/21/acm_triple_parity_raid/

5. Lori Alt: *ZFS as a Root Filesystem* http://hub.opensolaris.org/bin/download/Community+Group+zfs/boot/zfsboottalk.0910.pdf

6. Dave Pacheco: *Compression on the Sun Storage 7000*, 16.03.2009, http://blogs.sun.com/dap/entry/zfs_compression

7. Jeff Bonwick: *ZFS Deduplication*, 02.11.2009, http://blogs.sun.com/bonwick/entry/zfs_dedup

8. Franz Haberhauer: *Hybride Speicherarchitekturen mit ZFS - Transparente Nutzung von SSDs*, http://blogs.sun.com/solarium/entry/hybride_speicherarchitekturen_mit_zfs_und

9. Roch Bourbonnais: *When to (and not to) use RAID-Z*, 31.05.2006, http://blogs.sun.com/roch/entry/when_to_and_not_to

10. *Configuring Oracle Solaris ZFS for an Oracle Database*, Oracle White Paper, May 2010 http://developers.sun.com/solaris/docs/wp-oraclezfsconfig-0510_ds_ac2.pdf

11. *ZFS Evil Tuning Guide* http://www.solarisinternals.com/wiki/index.php/ZFS_Evil_Tuning_Guide

12. Sun Performance Technologies and ISV Engineering: Tuning: *ZFS for the F20 Card and F5100 Storage Array*, 22.04.2010, http://wikis.sun.com/display/Performance/Tuning+ZFS+for+the+F5100

13. *ZFS for Databases* http://www.solarisinternals.com/wiki/index.php/ZFS_for_Database

14. Official Blog of the Sun Microsystems Fishworks Engineering Team http://blogs.sun.com/fishworks

Kontaktadresse:

Franz Haberhauer
Oracle Deutschland B.V. & Co. KG
Zettachring 10a
D-70567 Stuttgart

Telefon: +49 (0) 72098-465
E-Mail Franz.Haberhauer@Oracle.com
Internet: www.oracle.de