Zur Nutzung und Verwaltung direkt oder über ein Storage-Area-Network (SAN) angebundener Platten für eine Oracle-Datenbank wird ASM auf Raw-/Block-Devices empfohlen, was von vornherein auf die I/O-Charakteristika der Datenbank abgestimmt ist. Nicht zuletzt aus administrativen Überlegungen heraus sind aber weiterhin auch Dateisysteme populär, wobei deren Voreinstellungen meist nicht den spezifischen I/O-Charakteristika von Datenbanken entsprechen, die sich ja von sonstigen Lastprofilen signifikant unterscheiden.

Best Practices für Datenbanken auf ZFS

Franz Haberhauer, ORACLE Deutschland B.V. & Co. KG

ZFS ist das moderne, innovative lokale Dateisystem von Oracle Solaris und kann als gängiges Dateisystem von Datenbanken genutzt werden. Darüber hinaus verfügt es über effiziente Snapshot- und Cloning-Funktionen. Andererseits dient es als internes Dateisystem der Network-Attached-Storage-Familie (NAS) von Oracle – der ZFS Storage Appliances (SA) – und wird damit indirekt für Datenbanken auf „(d)NFS“ genutzt. Der Artikel gibt Konfigurationshinweise und geht auf deren technische Hintergründe ein. Abschließend werden einige Tools angesprochen, die auf der ZFS SA verfügbar sind.

Oracle Solaris ZFS

Vor mittlerweile gut sieben Jahren wurde mit ZFS in Solaris 10 ein neues lokales Dateisystem integriert, das die klassischen Aufgaben eines Dateisystems – Daten ohne großen administrativen Aufwand schnell und effizient zu schreiben, sicher zu speichern und wieder auszulesen – mit einer innovativen Architektur grundsätzlich anders löste als traditionelle Dateisysteme [1]. Dateisystem und Volume-Management sind integriert, was die starken Mechanismen zur Sicherung der Datenintegrität ermöglicht, die ZFS besonders auszeichnen.

Stille Datenkorruption, teilweise durch transiente Hardware-Defekte oder Software-Fehler entstanden, ist ein reales Risiko, das angesichts der heutigen Datenvolumina kein unwahrscheinliches Ereignis mehr ist [2]. Über konsequent genutzte Prüfsummen in baumstrukturierten Dateien und Meta-Informationen kann ZFS korrupte Blöcke erkennen und gegebenenfalls aus vorhandenen redundanten Daten (Spiegel, RAID-Z, RAID-Z2) den korrekten Inhalt rekonstruieren. Die konsequente Nutzung des Copy-on-Write-Paradigmas und des Transaktionskonzepts sorgen dafür, dass das On-Disk-Image auch nach einem Systemabbruch immer konsistent ist, wobei Write-Caches moderner Platten und Speichersysteme genutzt werden können. ZFS erlaubt es zudem, relativ langsame Platten mit hoher Kapazität mit einem Level-2-Cache (L2ARC) aus schnelleren Laufwerken – insbesondere SSDs – für Anwendungen transparent zu leistungsfähigen „hybriden Storage Pools“ zu kombinieren. Leichtgewichtige Snapshots mit minimalem Overhead und der Möglichkeit, daraus beschreibbare Clones zu machen, Kompression und in Solaris 11 Verschlüsselung und Deduplikation sind nur einige der umfangreichen Funktionalitäten, die ZFS so attraktiv machen.

Als Hintergrund für die folgenden Empfehlungen zur Nutzung für Datenbanken ist von besonderem Interesse, wie ZFS Daten ausschreibt. ZFS überschreibt Daten nicht unmittelbar, sondern folgt dem Prinzip „Copy on write“. Geänderte Blöcke werden wie neue Blöcke an eine neue Stelle auf der Platte geschrieben. Dabei werden variable Blockgrößen von bis zu 128 KB verwendet und wahlfreie Schreibzugriffe in effiziente sequenzielle Schreiboperationen umgewandelt, was den Zugriffs-

Charakteristika moderner Platten entgegenkommt.

Änderungen werden von den Blättern zum Wurzelknoten, dem sogenannten „Uberblock“, propagiert. Das Ausschreiben des Uberblock bildet den atomaren 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, wobei die übergeordneten Knoten jeweils Prüfsummen für die darunterliegenden Knoten enthalten.

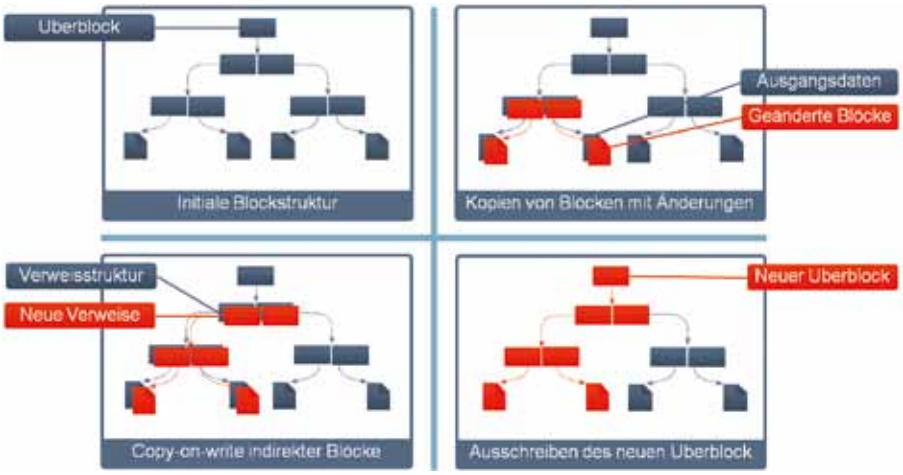

Abbildung 1: Copy-on-write: Update-Operation in ZFS

Währenddessen laufen weitere Änderungen in der nächsten Transaktionsgruppe. Wird der ersetzte Wurzelknoten nicht freigegeben, sondern sichtbar gemacht, erhält man mit minimalem Aufwand eine Snapshot-Funktionalität. Um synchrone Operationen, wie sie in DBMS oder von NFS-Servern genutzt werden, schnell abwickeln zu können, wird zudem ein Intent Log (ZIL) genutzt, über den Änderungen sofort auf stabilen Speichern gesichert werden, bevor sie sich mit dem Ausschreiben des Uberblock der Transaktionsgruppe auf den Datenplatten selbst manifestieren. Für eine optimale Performance kann man den ZIL von den Datenplatten separieren und auf dedizierten, besonders schnellen stabilen Speichern ablegen, wie etwa Flash-Karten oder SSDs, die zum Schutz vor Datenverlusten ebenfalls gespiegelt sein sollten.

Die I/O-Charakteristika der Oracle -Datenbank unterscheiden sich von denen sonstiger Lasten auf Dateisystemen. Während normale Dateien vom Dateisystem im Hauptspeicher gepuffert sind und – oft kleine – I/Os der Anwendungen vom Dateisystem asynchron beim Ausschreiben auf Platte in größere I/Os kumuliert oder beim Lesen vorausgelesen werden, puffert die Oracle-Datenbank Daten selbst in der SGA und schreibt sie bei Bedarf in einer definierten Blockgröße synchron aus. Dabei sind nicht alle I/Os gleichermaßen kritisch für die Performance. Die auf der Dateisystem-Ebene synchronen I/Os in die Daten- und Index-Bereiche sind aus Datenbank-Sicht asynchron, nur die Zeit für das Ausschreiben des Commit-Record in den Redo Log geht unmittelbar in die Antwortzeit einer Transaktion ein.

Bei älteren Dateisystemen wie Solaris UFS wurden einige Optimierungen insbesondere für Datenbanken in einer Option unter dem Begriff „Direkt I/O“ zusammengefasst, mit der Datenbank-Dateien optimiert geöffnet oder Dateisysteme gemountet werden können. Dadurch wird insbesondere beim UFS ein Single-Writer-Lock zur Sicherung der POSIX-Semantik, die für Datenbank-Dateien nicht relevant ist, zwecks Erhöhung des Durchsatzes umgangen und die (Doppel-)Pufferung im Dateisystem deaktiviert.

ZFS adressiert parallele I/Os POSIX-konform und performant durch Byte-Range-Locking. Die Pufferung im ZFS-Puffer des Hauptspeichers, dem ARC, kann gezielt durch ein Dataset-Property „primarycache“ mit den Werten „all“, „none“ oder „metadata“ gesteuert werden. Bei ZFS können in einem Storage Pool, der einem „logical Volume“ eines traditionellen Volume Manager entspricht, mehrere Datasets („Dateisysteme“) angelegt werden, die unterschiedliche Properties haben können.

Best Practices für Oracle-Datenbanken

Eine wesentliche Einstellung ist die Anpassung der ZFS „recordsize“ für Tabellen und Indizes an die Blockgröße der Datenbank („db_block_size, allerdings nicht kleiner als die Seitengröße des Servers, bei SPARC 8 KB). Für andere Bereiche (Redo Logs, Undo, Temp und Archived Redo Logs) passt hingegen die Voreinstellung von 128 KB. Um für diese Bereiche jeweils spezifische Parameter setzen zu können, sollten sie in separate Datasets gelegt werden.

„recordsize“ definiert man vorzugsweise gleich beim Anlegen eines Dataset, da eine Änderung nur für danach angelegte Dateien zum Tragen kommt. Um sie für eine bereits vorhandene Datei zu ändern, kann man diese kopieren. Durch Kopieren kann man auch einen Seiteneffekt von Copy-on-write wieder eliminieren: Wegen der wahlfreien Änderung einzelner Blöcke wird eine physische Nachbarschaft auf der Platte mit der Zeit aufgelöst, da ja geänderte Blöcke nicht an die ursprüngliche Stelle zurückgeschrieben, sondern an eine neue ausgeschrieben werden, was einerseits das Schreiben effizienter macht, andererseits bei nachfolgenden Tabellen-Scans dann zu wahlfreien statt sequenziellen Zugriffen führen kann, denn eventuell wird aus einem großen logischen Read der Datenbank eine Anzahl kleinerer IOs auf der ZFS-Ebene. Dieser Effekt macht sich auch bei Sicherungen über RMAN bemerkbar. Bei hohen Änderungsraten kann es angesichts der Laufzeit von Backups angebracht sein, auf „recordsize“ von 32 KB oder 64 KB zu gehen.

Ein weiterer Faktor, den man berücksichtigen sollte, kann eine minimale interne I/O-Größe von Speichersystemen sein. Diese resultiert meist aus der

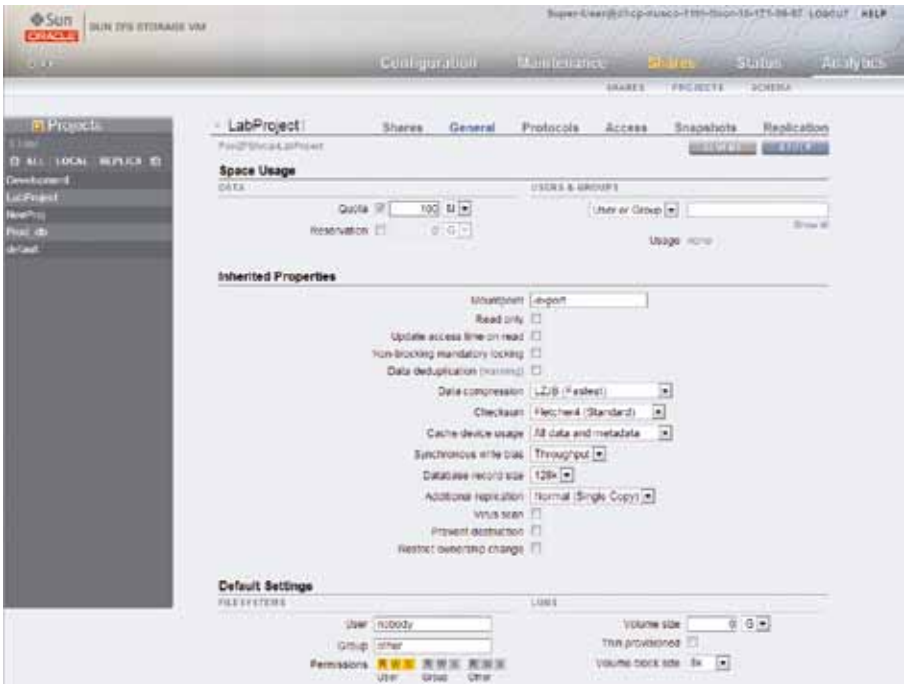

Abbildung 2: Web-GUI der ZFS Storage Appliance

internen Verwendung von RAID-Schemata. Generell ist unter Performance-Gesichtspunkten für die Datenbank die Spiegelung gegenüber RAID-5/-6 oder auch RAID-Z/-Z2 vorzuziehen – und zwar im Hinblick auf die stundenlangen Resilvering-/Wiederherstellungszeiten nach dem Ausfall einer heutigen Platte mit mehreren Terabyte Kapazität vorzugsweise mit mehrfacher Redundanz, um das Risiko eines weiteren Ausfalls in dieser Phase abzudecken.

Spiegel haben den Vorteil, dass durch sie die maximale Schreib-Rate (IOPS) erhalten bleibt und durch (Mehrfach-)Spiegel die maximale Lese-Rate entsprechend vervielfacht wird, was bei RAID-Verbünden nicht in gleichem Maße der Fall ist, wobei sich insbesondere der Aspekt des Schreibens bei Hardware-RAIDs mit nichtflüchtigen Caches (NVRAM) relativiert.

Bei Copy-on-write ist die Preallocation von Dateien nicht sinnvoll – ausreichend Platz lässt sich besser über Quotas und insbesondere Reservierungen im ZFS freihalten. Generell sollte für eine optimale Performance die Kapazität von Dateisystemen nicht voll ausgenutzt werden, da bei hohem Füllgrad auf Algorithmen umgeschwenkt wird, die Speichereffizienz über Performance stellen. Während traditionell für Datenbanken auf ZFS empfohlen wurde, unter 80 Prozent zu bleiben [3, 4], wurde für Solaris 11.1 nach Optimierungen diese Empfehlung auf 90 Prozent angehoben [5].

Zudem sollte für Anwendungen, die wie die Oracle-Datenbank sehr viel Hauptspeicher nutzen, der ZFS-Puffer (ARC) von vornherein entsprechend über den Parameter „zfs_arc_max im / etc/system“ beschränkt werden, um zu vermeiden, dass ZFS beim (Neu-)Start der Datenbank erst Speicher freigeben muss oder im Betrieb darum konkurriert. Wichtig ist, dass er ausreichend groß bleibt für die Metadaten des Workingsets (etwa 1,5 Prozent der aktiven Daten bei ZFS „recordsize“ von 8 KB, bei einer größeren entsprechend weniger). Hier ein Beispiel für die Beschränkung auf 2GB im „/etc/system“: §set zfs:zfs_arc_max=2147483648“ oder „set zfs:zfs_arc_max=0x80000000“. Für die Datasets der Undo-Daten und Archived Redo Logs kann man über „primarycache=metadata“ die Puffe-rung im ARC gleich auf die Metadaten beschränken.

Da Datenbank-I/Os für das Dateisystem synchron sind, kommt dem ZFS Intent Log (ZIL) eine besondere Bedeutung zu, weshalb er auf ein separiertes Log-Device aus (gespiegelten) SSDs gelegt werden sollte. Der ZIL kennt zwei Modi, die über den Parameter „logbias“ gesteuert werden. Während bei „latency“ (Voreinstellung) die Daten aus dem ARC zunächst in den ZIL und mit der Transaktionsgruppe auf die Datenplatten geschrieben werden, werden im Modus „throughput“ die Daten direkt auf die Datenplatten geschrieben und nur Verweise darauf in den ZIL (die dann mit dem Ausschreiben des neuen Uberblock obsolet werden). Da hier weniger in den ZIL geschrieben wird, lässt sich mit einer etwas höheren Latenz der einzelnen Writes ein größerer Gesamtdurchsatz erreichen – und zudem Bandbreite für wirklich Performance-kritische Writes bereitstellen, nämlich jene in die Redo-Logs. Insofern lautet die Empfehlung, für alle Bereiche außer den Redo Logs „logbias=throughput“ zu setzen.

Die automatische und transparente Kompression der Archived Redo Logs ist eine weitere Möglichkeit, spezifische Funktionalitäten von ZFS zu nutzen. Kompression spart übrigens nicht nur Plattenplatz, sie kann auch die Anzahl von I/Os reduzieren und sich darüber positiv auf die Performance auswirken. Kompression wird durch Setzen des Dataset-Property „compression=on“ aktiviert. In Tabelle 1 sind noch einmal die Empfehlungen tabellarisch zusammengefasst

Diese Empfehlungen, die in einem Best-Practices-Dokument auf „solarisinternals.com“ gesammelt wurden, haben inzwischen Eingang in das „So-

Dataset	**ZFS recordsize**	**logbias**	**primarycache**
Daten	**db_block_size*)**	**throughput**	all
Indizes	**db_block_size**	**throughput**	all
Redo	128KB	latency	all
Undo	**db_block_size*)**	**throughput**	**metadata**
Temp	128KB	**throughput**	all
Archive	128KB **compression=on**	**throughput**	**metadata**

*) Datawarehouse-artige Anwendungen, Large Objects (LOBs): 128KB (Default)

Tabelle 1: Abweichungen von den Voreinstellungen sind fett dargestellt

laris Tunable Parameters Reference Manual“ [4, 5] und in ein im September 2012 aktualisiertes Whitepaper [3] gefunden, die jeweils ausführliche Beispiele enthalten. [3] enthält zudem einen Abschnitt mit Empfehlungen zur Formatierung von LUNs und Hinweisen zur Nutzung von Caches in Platten und Plattensystemen, die für ZFS ganz allgemein gelten.

MySQL

Für MySQL wird ebenfalls empfohlen, „recordsize“ an die Blockgröße der Storage Engine anzupassen, bei InnoDB 16 KB für die Data Files und die Voreinstellung von 128 KB für die Log Files, wobei Data und Log Files in separate Pools gelegt werden sollten [4, 5].

ZFS Storage Appliance

Auch bei der Nutzung von Datenbanken über „(d)NFS“ spielt ZFS unter Umständen eine Rolle, nämlich als internes Dateisystem der ZFS Storage Appliances (SA), einer Produktlinie leistungsfähiger NAS-Systeme mit einem Einstiegssystem mit elf Platten bis hin zu hochverfügbaren Konfigurationen und einer Kapazität von mehreren Petabyte, jeweils mit integriertem Flash/SSDs für separierte ZFS Intent Logs [6]. Die Systeme zeichnen sich durch ein im Vergleich zum Wettbewerb sehr gutes Preis-Leistungs-Verhältnis aus sowie durch eine intuitive, webbasierte Administrationsoberfläche, in der sich die oben erläuterten Parameter (ZFS Properties) wiederfinden – teilweise unter logischen Bezeichnungen: „logbias“ als „Synchronous write bias“ oder „recordsize“ als „Database record size“ (siehe Abbildung 2).

Eine weitere interessante Funktionalität der ZFS SA ist Analytics. Sie erlaubt tiefgehende Performance-Analysen – sehr fein granuliert bis hin zu Offsets in einzelnen Dateien und mit einer grafischen Visualisierung über der Zeitachse (siehe Abbildung 3).

Die ZFS SA wird auch als integrierter Fileserver in den Engineered Systems Exalogic und SPARC SuperCluster verbaut und ist als ZFS Backup Appliance als leistungsstarke Backup-Lösung für die Engineered Systems verfügbar. Zwei Whitepaper [7, 8] beschreiben ausführlich Best Practices der Nutzung

Abbildung 3: I/O-Profil eines „CREATE TABLESPACE” in ZFS Storage Appliance Analytics

von ZFS SA für Backup und Recovery der Exadata, wobei vieles davon auf die Sicherung von Datenbanken auf anderen Servern übertragbar ist.

Wie gesagt sind mit ZFS sehr effiziente Snapshots und Clones möglich, was sich hervorragend nutzen lässt, um etwa schnell und platzsparend Entwicklungs- oder Test-Datenbanken anzulegen [9]. Mit dem neuen Snap Management Utility [10] kann das Erzeugen von Datenbank-Snapshots und -Clones in Verbindung mit ZFS Storage Appliances weitgehend automatisiert und über eine webbasierte Oberfläche verwaltet werden. Weitere Einsatzfelder ergeben sich daraus, dass auf der ZFS SA Hybrid Columnar Compression (HCC) auch außerhalb der Exadata nutzbar ist.

Allein in der Oracle IT und in den Cloud Datacentern werden inzwischen ZFS Storage Appliances mit einer Gesamtkapazität von über 200 Petabyte eingesetzt. In [11] sind daraus Anwendungsszenarien beschrieben, in denen Best Practices exemplarisch umgesetzt sind.

Literaturhinweise

[1] Oracle Solaris ZFS Technology: http://www.oracle.com/technetwork/server-storage/solaris11/technologies/zfs-338092.html

[2] Robin Harris: CERN's Data Corruption Research, 17.09.2007: http://storagemojo.com/2007/09/19/cerns-data-corruption-research

[3] Cyndi Swearingen, Roch Bourbonnais, Alain Chéreau: Configuring ZFS for an Oracle Database, Oracle White Paper, September 2012: http://www.oracle.com/technetwork/server-storage/solaris10/config-solaris-zfs-wp-167894.pdf

[4] Oracle Solaris 10 1/13 1 Tunable Parameters Reference Manual – Tuning ZFS for Database Products: http://docs.oracle.com/cd/E26505_01/html/E37386/chapterzfs-db1.html#scrolltoc

[5] Oracle Solaris 11.1 Tunable Parameters Reference Manual – Tuning ZFS for Database Products: http://docs.oracle.com/cd/E26502_01/html/E29022/chapterzfs-db1.html#scrolltoc

[6] Oracle Sun ZFS Storage Appliances: http://www.oracle.com/us/products/servers-storage/storage/nas

[7] Protecting Oracle Exadata with the Sun ZFS Storage Appliance: Configuration Best Practices, Updated Oracle White Paper, March 2013: http://www.oracle.com/technetwork/server-storage/sun-unified-storage/documentation/zfssa-exadata-rman-v1-3-1926901.pdf

[8] Backup and Recovery Performance and Best Practices using the Sun ZFS Storage Appliance with the Oracle Exadata Database Machine, Oracle White Paper April 2012: http://www.oracle.com/technetwork/database/features/availability/maa-wp-dbm-zfs-backup-1593252.pdf

[9] Database Cloning using Oracle Sun ZFS Storage Appliance and Oracle Data Guard, Oracle White Paper, December 2011: http://www.oracle.com/technetwork/database/features/availability/maa-db-clone-szfssa-172997.pdf

[10] Snap Management Utility for Oracle Database: http://www.oracle.com/us/products/servers-storage/storage/nas/snap

[11] Sun ZFS Storage Appliance and Oracle IT: Use Cases and Benefits, Oracle White Paper, September 2012: http://www.oracle.com/us/products/servers-storage/storage/nas/resources/zfssaoracle-it-whitepaper-100812gc-1875031.pdf

Franz Haberhauer
franz.haberhauer@oracle.com