Neue ZFS-Funktionen in Oracle Solaris 11

Constantin Gonzalez
Oracle Deutschland B.V. & Co. KG
München

Schlüsselworte:

Oracle Solaris 11, Solaris, ZFS, De-Duplikation, Verschlüsselung, Filesystem, Dateisystem, Performance.

Einleitung

Oracle Solaris 11 bringt neue Funktionen im ZFS Filesystem: Das root-Filesystem ist jetzt ein ZFS-Filesystem in einem Root-Pool mit weitreichenden Vorteilen für Installation und Update des OS. De-Duplikation spart Platz durch geschickte Erkennung und Vermeidung doppelter Datenblöcke. Schließlich erlaubt Encryption die Verschlüsselung von Filesystemen und Volumes. Darüber hinaus gibt es zahlreiche Verbesserungen in Robustheit, Performance, der Fehlerbehandlung, und der Nutzerfreundlichkeit.

Im Zentrum von Solaris 11

Das ZFS Dateisystem wurde 2006 in einem Update von Solaris 10 eingeführt und damit gegenüber dem Solaris 10 Betriebssystem quasi "nachgereicht". Seitdem erfreut sich ZFS einer stetig steigenden Beliebtheit, die neben Oracle Solaris auch andere Ableger von Solaris und andere Betriebssysteme wie FreeBSD bis hin zu Linux erfaßt hat. ZFS-Nutzer können heute bereits auf über ein halbes Jahrzehnt Erfahrung mit dem Dateisystem zurückblicken.
Mit Solaris 11 ist ZFS nun in das Zentrum des Betriebssystems vorgerückt: Hier wird das gesamte Betriebssystem standardmäßig in einen Zpool installiert, wobei Snapshots und Clones die Grundlage für Boot Environments bilden, wichtige Bestandteile des mit Solaris 11 neu eingeführten Installations- und Package-Management-Systems. Hinzu kommen zwei zentrale neue Funktionen: De-Duplikation und Verschlüsselung, die Integration in andere zentrale Services von Solaris 11 wie den CIFS-Service oder COMSTAR, und zahlreiche Verbesserungen der Performance, Anwenderfreundlichkeit sowie natürlich Fehlerbehebungen.

ZFS als Root-Filesystem

Beginnend mit Solaris 11 ist ZFS das Standard-Filesystem für die System-Installation, also das Root-Verzeichnis. UFS oder andere Filesysteme sind für das System-Verzeichnis nicht vorgesehen. Dies hat folgende Vorteile:

- Die Betriebssystem-Installation profitiert von der Daten-Integrität, die ZFS bietet. Natürlich können mit ZFS genauso Boot-Platten gespiegelt werden wie vorher, allerdings funktioniert das jetzt schneller und zuverlässiger, als bei traditionell gespiegelten Boot-Platten.
- Snapshots und Clones können als mächtigeren, schnelleren und komfortableren Ersatz für Live Upgrade genutzt werden.
- Beliebig viele Varianten des Betriebssystems können ebenfalls durch Clones gleichzeitig vorgehalten werden und verwenden dabei nur minimal zusätzlichen Platz.
- Mit etwas Geschick kann auch Kompression für das Root-Verzeichnis verwendet werden.

Ein Boot-fähiges ZFS-Filesystem wird in Solaris 11 ein *Boot Environment* genannt. Beim Booten wird im GRUB Bootloader eines von mehreren Boot-Environments ausgewählt. Boot Environments können geklont werden, was automatisch bei der Installation von Updates passiert. Dabei wird der alte Stand des Betriebssystem als Boot Environment beibehalten und die Installation des Updates kann auf einem Klon des aktuellen Boot Environments während der Laufzeit des Betriebssystems erfolgen. Erst durch Booten des aktualisierten Boot Environments wird der Update aktiv und so beschränkt sich eine durch einen Update des OS bedingte Ausfallzeit des System lediglich auf die Zeit eines Boot-Vorganges. Das „alte“ Boot-Environment bleibt dabei als Rückfalloption erhalten.

Das Zusammenspiel zwischen dem Bootloader Grub, ZFS, dem Root-Filesystem und dem IPS Package Management System wird durch das Kommando `beadm(1M)` verwaltet. Damit kann der Administrator jederzeit eigene Boot Environments erzeugen, etwa um getestete und abgenommene Konfigurationen des Systems oder Installations-Zwischenstände jederzeit zugreifbar und bootfähig zu archivieren. Boot Environments lassen sich selbstverständlich auch beliebig mounten, umbenennen und löschen.

De-Duplikation in ZFS

Unter Speicherherstellern ist De-Duplikation zum echten Mode-Feature geworden. Hierbei geht es darum, identische Datenbereiche zu erkennen und nur einfach zu speichern, so dass zufällig oder absichtlich doppelt gespeicherte Daten keinen zusätzlichen Speicherplatz erfordern. Je nach Anwendung kann die Speicherersparnis zwischen einem paar Prozent (etwa in einem typischen Home-Verzeichnis eines Users) und einem vielfachen der gesamten Speicher-Infrastruktur liegen (z.B. in einer Virtualisierungs-Farm mit zahlreichen ähnlich konfigurierten virtuellen Maschinen).

De-Duplikation in ZFS ist:

- **Block-Basiert**: Jeder abgespeicherte Block wird in ZFS auf Duplizität hin geprüft, auch wenn sich doppelte Daten in verschiedenen und verschieden langen Dateien befinden. Dabei gilt die dynamische Block-Größe von ZFS sowie die Ausrichtung an Block-Grenzen.
- **Synchron**: Blöcke werden während des Speichervorganges auf ihre Duplizität überprüft und doppelte Blöcke sofort entfernt. Es gibt keinen Hintergrundprozess, der nachträglich Daten nach doppelten Blöcken durchkämmt.
- **Prüfsummen-basiert**: Blöcke werden anhand ihrer ZFS-Prüfsumme verglichen. Zur Vermeidung von falsch-positiven Vergleichen wird daher bei Verwendung von De-Duplikation auch die Verwendung von SHA-256-Prüfsummen empfohlen (das ist auch die Standard-Einstellung). Diese Methode ist sehr sicher: Die Wahrscheinlichkeit einer Hash-Kollision beim SHA-256-Algorithmus ist 2^{-256} bzw. ca. 10^{-77}. Dies ist 50 Größenordnungen weniger wahrscheinlich als eine unentdeckter, nicht korrigierter ECC-Speicherfehler.[1] Dennoch bietet ZFS auch an, Blöcke 1:1 zu prüfen, wenn ihre Prüfsummen identisch sind, um etwa bei Wahl eines schwächeren Hash-Algorithmus trotzdem Sicherheit zu haben.
- **Dynamisch administrierbar**: De-Duplikation kann in ZFS jederzeit ein- und ausgeschaltet werden, sowie der Prüfsummen-Algorithmus und die Art der Prüfung jederzeit verändert werden. Dabei wirkt sich (ähnlich wie bei Kompression) eine Veränderung der Einstellung immer auf danach gespeicherte Daten aus: Vor der Aktivierung von De-Duplikation gespeicherte Daten werden bei der De-Duplikation nicht berücksichtigt. Bereits de-duplizierte Daten bleiben auch nach Deaktivierung von De-Duplikation weiter De-Dupliziert. Standard-mäßig ist De-Duplikation nicht aktiv.

1 Jeff Bonwick: ZFS Deduplication, Blog-Eintrag, http://blogs.oracle.com/bonwick/entry/zfs_dedup

- **Kompatibel**: ZFS De-Duplikation arbeitet mit anderen ZFS-Funktionen wie Kompression oder Verschlüsselung nahtlos zusammen. Nutzt man alle diese Funktionen gemeinsam, so gilt die Reihenfolge Kompression->Verschlüsselung->De-Duplikation.

ZFS De-Duplikation in der Praxis

De-Duplikation wird in ZFS mit der `dedup`-Eigenschaft auf Dateisystemebene dynamisch eingestellt:

```
# zfs set dedup=on tank/home
```

ZFS pflegt eine Pool-weite Dedup-Tabelle, die alle Blöcke mit Adresse und Prüfsumme referenziert, die nach Aktivierung von De-Duplikation geschrieben wurden. Die Einstellung von De-Duplikation wird also auf Filesystem-Ebene gepflegt, die Implementation geschieht jedoch auf Pool-Ebene. Jedes mal wenn ein neuer Daten-Block bei aktiver De-Duplikation geschrieben wird, prüft ZFS anhand der Dedup-Tabelle nach, ob schon mal ein Block mit der gleichen Prüfsumme geschrieben wurde. Wird der zu schreibende Block nach dieser Prüfung (und einer evtl. weiteren 1:1-Prüfung) als Doublette erkannt, wird er verworfen und statt dessen ein Verweis auf den bereits vorhandenen Block für den neuen Block angelegt. Ist der Block neu, wird er regulär geschrieben und ein neuer Eintrag in der Dedup-Tabelle angelegt.

ZFS kann den Erfolg der De-Duplikation anzeigen:

```
# zpool list tank
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
tank 136G 55.2G 80.8G 40% 2.30x ONLINE -
```

Für Speicherplatz-abhängige Funktionen wie Quotas oder den angezeigten Verbrauch gelten immer die Speichermengen der nicht de-duplizierten Daten, während die Pool-Statistiken wie oben im Beispiel stets den tatsächlichen Speicher-Verbrauch anzeigen.

Ebenso einfach wie die Aktivierung erfolgt die De-Aktivierung von De-Duplikation:

```
# zfs set dedup=off tank/home
```

Danach bleiben bereits de-duplizierte Daten wie sie sind, jedoch werden neue Daten nicht mehr de-dupliziert.

Wer auf Nummer sicher gehen will, kann eine 1:1-Prüfung von Blöcken nach erfolgreichem Prüfsummen-Vergleich einstellen:

```
# zfs set dedup=verify tank/home
```

Ebenso leicht kann man auch den Prüfsummen-Algorithmus für die De-Duplikation wechseln. Standardmäßig wird `SHA256` verwendet. Das ist ein besonders sicheres Verfahren, bei dem kein `verify` nötig ist. Man kann aber z.B. ein schwächeres Verfahren wählen, das weniger Rechenzeit erfordert, und sich gleichzeitig durch die `verify`-Option absichern:

```
# zfs set dedup=fletcher4,verify tank/home
```

Vor- und Nachteile von De-Duplikation in ZFS

De-Duplikation spart nicht nur Platz, sondern führt auch in der Regel zu besserer Performance: Bereits geschriebene Blöcke müssen nicht nochmal geschrieben werden und sparen daher I/O-Bandbreite. Ebenso können bereits gelesene Blöcke quasi „recycled“ werden, was Lese-Bandbreite spart.

Es gibt jedoch einen Preis, den man für die Duplikation bezahlt und das ist der Platz für die Dedup-Tabelle. Diese wird persistent als Teil der Metadaten im Pool gespeichert und im Hauptspeicher des Rechners gepuffert. Im Interesse von optimaler Performance sollte man darauf achten, daß der Hauptspeicher groß genug dimensioniert ist, um die gesamte Dedup-Tabelle aufzunehmen, bzw. den Hauptspeicher durch Flash Memory oder SSDs um eine zweite Stufe (L2ARC) erweitern, damit Dedup-Tabellen-Zugriffe schnell genug erfolgen können und nicht zu übermäßigen Festplattenzugriffen führen.

Für die Berechnung des Speicherbedarfs für die Dedup-Tabelle gibt es ein paar Regeln:

- Jeder Eintrag in der Dedup-Tabelle „kostet“ etwa 320 Byte[2] pro Block. Die Größe der Dedup-Tabelle hängt also von der *Anzahl* der ZFS-Datenblöcke ab, nicht notwendigerweise von der *Menge* der Daten, da ZFS eine variable Blocklänge verwendet.
- Eine Abschätzung der Anzahl von Datenblöcken in einem Pool kann mit dem Befehl `zdb -b <poolname>` gemacht werden, oder man nimmt eine mittlere Block-Größe an (64K) und teilt die Datenmenge durch die Blockgröße, um auf einen Wert für die Anzahl der Blöcke zu kommen.
- ZFS verwendet maximal den gesamten Hauptspeicher der Maschine minus 1GB als ARC-Cache (7/8 des Hauptspeichers bei Maschinen mit weniger als 1 GB Hauptspeicher). Standardmäßig darf nur 1/4 des ARC für Metadaten (wie etwa die Dedup-Tabelle) verwendet werden, so daß der Hauptspeicherbedarf einer Maschine bei mindestens 4 x Dedup-Größe + 1 GB liegt, wenn die Dedup-Tabelle ganz im Hauptspeicher liegen soll.

Als Faustregeln lassen sich daraus ableiten:

- Pro TB Pool-Daten werden bei einer mittleren Block-Größe von 64K etwa 5GB für die Dedup-Tabelle gebraucht.
- Das bedeutet, dass man pro TB Pool-Daten mit 20 GB Hauptspeicher (oder Hauptspeicher + SSD für L2ARC) rechnen sollte + 1 GB für das Betriebssystem, um keine oder nur geringe Schreib-Performance-Einbußen durch den Zugriff auf die Dedup-Tabelle erwarten zu können.

Dem Aufwand für De-Duplikation muß natürlich ein Nutzen gegenüberstehen und der läßt sich bei einem vorhandenen Pool mit repräsentativen Daten mit folgendem Befehl abschätzen:

```
# zdb -S <poolname>
Simulated DDT histogram:

bucket              allocated                       referenced
______   ______________________________   ______________________________
refcnt   blocks   LSIZE   PSIZE   DSIZE   blocks   LSIZE   PSIZE   DSIZE
------   ------   -----   -----   -----   ------   -----   -----   -----
     2      623   69.9M   69.9M   69.9M    1.83K    210M    210M    210M
     4       14   1.63M   1.63M   1.63M       84    9.8M    9.8M    9.8M
 Total      637   71.5M   71.5M   71.5M    1.91K    219M    219M    219M

dedup = 3.07, compress = 1.00, copies = 1.00, dedup * compress / copies = 3.07
```

2 Siehe ZFS Dedup FAQ: http://hub.opensolaris.org/bin/view/Community+Group+zfs/dedup

Der Befehl gibt den erwarteten Dedup-Faktor aus (hier im Beispiel 3,07). Das ist die Menge an gespeicherten Daten, geteilt durch die Menge an dafür benötigtem Speicherplatz. In diesem Beispiel konnten also dank De-Duplikation drei mal mehr Daten gespeichert werden, als tatsächlich an Speicherplatz verbraucht wurde, oder anders herum gesagt konnte etwa 67% (1 – 1/Faktor) des Speicherplatzes eingespart werden.

Abschließend läßt sich also festhalten, daß die Verwendung von De-Duplikation in ZFS durchaus enorme Speichereinsparungen versprechen kann (z.B. bei virtualisierten Umgebungen), dafür aber in genug Hauptspeicher und ggfs. SSDs für L2ARC investiert werden sollte, damit Schreib-Zugriffe optimal schnell erfolgen können. Dies führt je nach Kosten für Hauptspeicher, SSDs und Festplatten-speicher in der Regel zu einem Break-Even-Punkt für das Verhältnis von Doubletten zu einzigartigen Daten, ab dem sich die Investitionen in RAM und/oder SSDs für De-Duplikation lohnen.

Hier gelten die Faustregeln:

- Ist die erwartete Einsparung durch De-Duplikation eher im unteren zweistelligen Prozent-Bereich, dann lohnt sich der Aufwand für De-Duplikation in der Regel nicht, weil der Preis für den eingesparten Festplattenplatz kleiner ist, als der Preis für den zusätzlichen RAM und Flash Memory-Aufwand.
- Ist die erwartete Einsparung größer als 50%, dann lohnt es sich in der Regel, die Maschine mit zusätzlichem Hauptspeicher und ggfs. Flash Memory für L2ARC auszurüsten, um das Einspar-Potenzial durch De-Duplikation auszuschöpfen.

Eine genauere Diskussion der Speicher-Anforderungen und Performance-Auswirkungen von ZFS De-Duplikation ist im Blog des Autors verfügbar.[3]

ZFS Verschlüsselung

Schon kurz seit der Einführung von ZFS wurde der Ruf nach einer Verschlüsselungs-Möglichkeit laut. Diese ist jetzt mit Oracle Solaris 11 verfügbar und sie arbeitet harmonisch mit anderen Security-Komponenten von Solaris zusammen.

ZFS Verschlüsselung bedeutet, dass jeder Datenblock mit einem Schlüssel verschlüsselt wird, der von Solaris verwaltet wird. Dabei wird zwischen einem „Data Encryption Key“ und einem „Wrapping Key“ unterschieden: Ersterer ist der Schlüssel, der bei der Verschlüsselung der Daten benutzt wird. Data Encryption Keys werden vom System basierend auf Zufallszahlen bei Bedarf automatisch erzeugt. Alle solchen Data Encryption Keys für ein Filesystem werden mit Hilfe des Wrapping Keys nochmal verschlüsselt. Der Wrapping Key ist der Schlüssel, der durch den Benutzer frei gewählt werden kann. Dieser Mechanismus ermöglicht es, dass der Benutzer seinen Schlüssel jederzeit ändern kann, ohne dass alle Daten neu verschlüsselt werden müssen.

ZFS Verschlüsselung in der Praxis

Will man für ein ZFS-Filesystem Verschlüsselung aktivieren, muss man das schon bei der Erzeugung angeben. Nachträglich läßt sich Verschlüsselung nicht aktivieren:

```
# zfs create -o encryption=on tank/home/constant
Enter passphrase for 'tank/home/constant': xxxxxxxx
Enter again: xxxxxxx
```

3 http://constantin.glez.de/blog/2011/07/zfs-dedupe-or-not-dedupe

Die Einstellung `encryption=on` führt zur Verwendung eines Standard-Verschlüsselungs-Algorithmus, der zur Zeit `aes-128-ccm` ist.

In diesem Beispiel wurde der Wrapping Key vom Benutzer auf der Kommandozeile eingegeben. Alternativ kann der Wrapping Key auch aus einer Datei stammen, wo der Schlüssel binär oder als Hex-Code vorgehalten wird:

```
# pktool genkey keystore=file outkey=/dmkey.file keytype=aes
keylen=256
# zfs create -o encryption=aes-256-ccm -o
keysource=raw,file:///dmkey.file tank/home/constant
```

In diesem Beispiel wurde mit `pktool` ein neuer Schlüssel mit einem anderen Algorithmus (`aes-256-ccm`) generiert und im zweiten Schritt ZFS für die Erzeugung des neuen Filesystems mitgegeben.

Die Verschlüsselungs-Eigenschaften eines Filesystems werden auch an seine Nachkommen wie Snapshots und Clones vererbt.

Will man den Schlüssel verändern, muss vorher der alte Schlüssel von ZFS geladen worden sein. Das geschieht entweder mit `zfs key -l` oder durch Mounten des Filesystems (z.B. auch beim Booten). Danach kann der Schlüssel gewechselt werden:

```
# zfs key -c tank/home/constant
Enter new passphrase for 'tank/home/constant': xxxxxxxx
Enter again: xxxxxxxx
```

Der Data Key ist nie sichtbar, kann aber bei Bedarf neu erzeugt werden:

```
zfs key -K tank/home/constant
```

Dabei werden jedoch nur neu gespeicherte Daten mit dem neuen Schlüssel verschlüsselt. Bereits vorher gespeicherte Daten bleiben mit ihrem alten Schlüssel gespeichert. ZFS merkt sich alle Data Keys (verschlüsselt über den Wrapping Key) um bei Bedarf immer alle Daten entschlüsseln zu können.

Randbedingungen für die ZFS-Verschlüsselung

Bei Verwendung von Verschlüsselung mit ZFS gibt es ein paar Bedingungen:

- Verschlüsselte ZFS-Filesysteme sind nicht boot-fähig.
- Ein mit `zfs send` gesendeter, nicht verschlüsselter ZFS Datenstrom kann nicht auf einem verschlüsselten ZFS-Filesystem empfangen werden. Hier muss man alternative Mechanismen wie `cp -r`, `find | cpio`, `tar`, oder `rsync` verwenden, um nicht verschlüsselte Daten in ein verschlüsseltes Filesystem zu migrieren.
- Es ist möglich, einen verschlüsselten Datenstrom mit `zfs send | zfs receive` zu übertragen, jedoch wird er dann mit einem neu erzeugten Schlüssel gespeichert und ist während der Übertragung nicht verschlüsselt.

Spezielle Eigenschaften für verschlüsselte ZFS Filesysteme

Neben der bereits bekannten `encryption`-Eigenschaft gibt es noch zwei weitere interessante Eigenschaften von ZFS Filesystemen:

- Die `keysource`-Eigenschaft gibt das Format und die Herkunft des Wrapping Keys an:

```
# zfs get keysource tank/home/constant
NAME                PROPERTY   VALUE              SOURCE
tank/home/constant  keysource  passphrase,prompt  local
```

- Die `rekeydate`-Eigenschaft gibt an, wann der Data Key das letzte mal geändert wurde:

```
# zfs get rekeydate,creation oracle
NAME                PROPERTY   VALUE                   SOURCE
tank/home/constant  rekeydate  Wed Nov 17 10:14 2010   local
tank/home/constant  creation   Wed Nov 17 10:14 2010   -
```

  Ist der Wert für `rekeydate` gleich dem Wert für `creation`, dann wurde der Data Key für das betroffene Filesystem noch nie geändert.

Weitere Informationen über ZFS-Verschlüsselungen finden sich im Blog von Darren Moffat:

- *Introducing ZFS Crypto in Oracle Solaris 11 Express*, http://blogs.oracle.com/darren/entry/introducing_zfs_crypto_in_oracle
- Having my secured cake and Cloning it too (aka Encryption + Dedup with ZFS), http://blogs.oracle.com/darren/entry/compress_encrypt_checksum_deduplicate_with
- ZFS encryption what is on disk?, http://blogs.oracle.com/darren/entry/zfs_encryption_what_is_on

Weitere Verbesserungen im ZFS Filesystem

Neben den drei großen Themen Root-Filesystem, De-Duplikation und Verschlüsselung sind in Solaris 11 noch zahlreiche Detailverbesserungen in ZFS eingeflossen. Hier ein Auszug der wichtigsten neuen Punkte:

- **Importieren problematischer Pools**: ZFS kann jetzt besser Pools importieren, die problematisch sind, weil sie z.B. kein Log-Device mehr haben (`zpool import -m`) oder weil die Schreib-Reihenfolge durch den Massenspeicher nicht eingehalten wurde und der Schreib-Vorgang unterbrochen wurde (`zpool import -F`). Im ersten Fall gehen Transaktionen verloren, die nur in den Log-Devices aber nicht im Pool vorhanden sind und der Import fällt auf die letzte gültige Transaktion zurück. Im zweiten Fall sucht ZFS im Pool nach dem letzten gültigen Transaktions-Stand und fällt darauf zurück. Alle unvollständigen Transaktionen, die danach initiiert wurden gehen verloren.
  Sollten diese Maßnahmen nicht helfen, gibt es jetzt auch die Möglichkeit, einen Pool im read-only-Modus zu importieren, um Daten daraus zu retten:
  `zpool import -o readonly=on tank`.
- **Verbesserte Pool-Meldungen**: Bei `zpool scrub`-Operationen und beim Resilvering von Pools werden jetzt noch detailliertere Meldungen und Statistiken angezeigt. Abschlußmeldungen für Scrubs und Resilver-Operationen bleiben über einen Reboot hinaus erhalten. Das `zpool status`-Kommando kann jetzt Daten in Intervallen (ähnlich z.B.: `iostat 5`) ausgeben.

- **Verbesserungen bei `zfs send/receive`**: Eigenschaften von ZFS Snapshots können während der Übertragung mit `zfs send/receive` verändert werden. Dabei kann auch angegeben werden, dass Eigenschaften aus dem Original übernommen, oder beim Empfang weggelassen werden sollen. Dies ist z.B. nützlich bei der Erzeugung von komprimierten Backups von nicht komprimierten Filesystemen.
- **Unterschiede zwischen ZFS Snapshots**: Mit dem neuen Sub-Befehl `zfs diff` können jetzt ZFS Snapshots miteinander verglichen werden:

```
$ ls /tank/home/constant
fileA
$ zfs snapshot tank/home/constant@old
$ ls /tank/home/constant
fileA fileB
$ zfs snapshot tank/home/constant@new
$ zfs diff tank/home/constant@old tank/home/constant@new
M /tank/home/constant/
+ /tank/home/constant/fileB
```

  Dabei bedeutet „`M`“, dass die Datei/das Verzeichnis gegenüber dem ersten Snapshot modifiziert wurde, „`+`“ bzw. „`-`“, dass die Datei/das Verzeichnis gegenüber dem ersten Snapshot hinzugefügt bzw. gelöscht wurde und „`R`“ steht für „Umbenennung“ („Rename“).

ZFS Performance-Verbesserungen

Auch die Performance von ZFS wurde an verschiedenen Stellen verbessert:

- Die Verwaltung und Auswahl von freien Blöcken wurde überarbeitet und optimiert. Dadurch wird das Schreiben neuer Daten beschleunigt.
- Beim Scrubbing von Pools wird Prefetching verwendet, um den Vorgang zu beschleunigen.
- Beim Scrubbing/Resilvering werden nur Rohdaten verwendet, also Blöcke nicht entschlüsselt oder entpackt. Dies spart Rechenzeit.
- NFS und CIFS verwenden Zero-Copy-I/O: Über alle OS-Ebenen hinweg wird derselbe Datenpuffer verwendet, ohne dass Daten umkopiert werden müssen.
- Der Synchronisations-Modus kann eingestellt werden, um auf Kosten von Datenintegrität mehr Performance erreichen zu können.
- RAID-Z-Pools können stellenweise bestimmte Daten auch spiegeln, um die Performance zu verbessern („hybrid allocation“).

Zusammenfassung

Solaris 11 liefert eine Menge von wesentlichen Verbesserungen im ZFS Filesystem aus. Zentrale neue Funktionen sind die Rolle von ZFS als Root-Filesystem im Zusammenspiel mit Installation, Paketverwaltung und Boot Environments, sowie die neuen Funktionen zur De-Duplizierung und Verschlüsselung von Filesystemen. Daraus ergeben sich zahlreiche neue Möglichkeiten für Systemadministratoren und System-Architekten.

Darüber hinaus bietet ZFS in Solaris 11 zahlreiche Detailverbesserungen, die die Stabilität, Datenintegrität, Performance und Nutzerfreundlichkeit deutlich verbessern.

Kontaktadresse:

Constantin Gonzalez
Oracle Deutschland B.V. & Co. KG
Riesstraße 25
D-80992 München

Telefon:	+49 (0) 89-14 30-25 40
E-Mail	constantin.gonzalez@oracle.com
Blog:	http://constantin.glez.de
Twitter:	@zalez