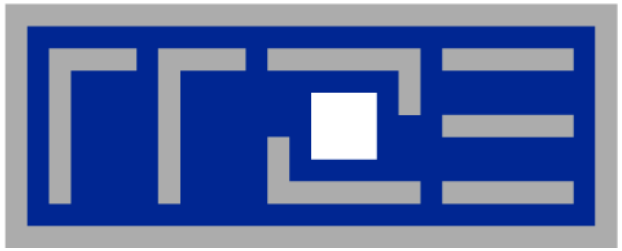


Solaris 10 - das ZFS

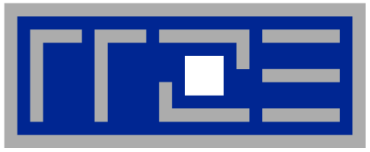
Gregor Longariva

16. Mai 2006

Überblick

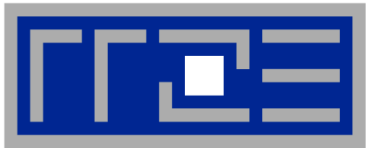
- Einleitung – das wichtigste in Kürze
- Überblick
- ZFS im Vergleich mit dem klassischen Konzept
- ZFS – wie wird die Datenintegrität garantiert
- Snapshots/Clones
- Weitere Besonderheiten
- Wenn noch Zeit ist: Beispiele

ZFS – das wichtigste in Kürze



- Maximale Datenintegrität
- Zukunftssicher:
  - 128 bit (2^128 = 3,402823669e+38 Blöcke)
  - Andere aktuelle Dateisysteme: 64 bit
- Administration sehr einfach
- Hohe Performance

ZFS - Überblick



Vollkommen neues Konzept:

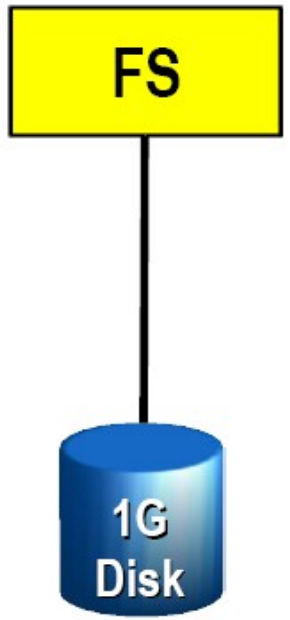
ZFS - Überblick

Vollkommen neues Konzept:

- **Es gibt keine Partitionen mehr**
- **Es gibt keine Volumes mehr**

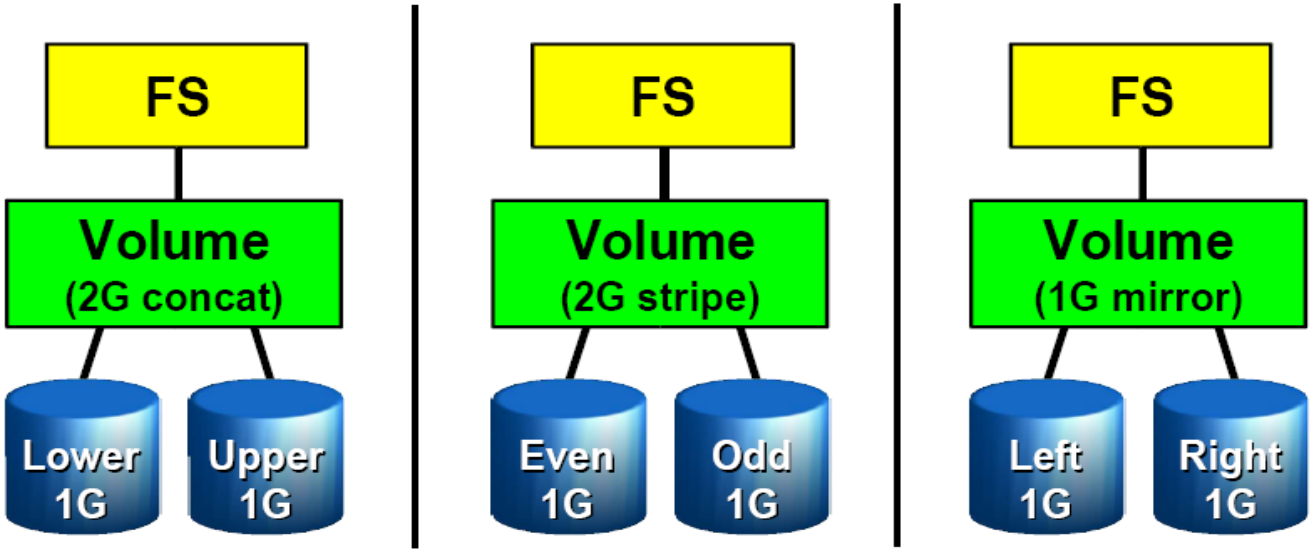
ZFS im Vergleich

- **Klassisches Modell:**
  - **Zuerst: Filesystem liegt auf einem Slice oder einer Platte**

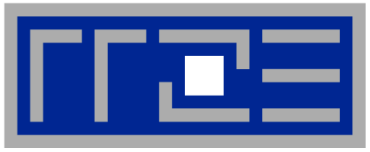


ZFS im Vergleich

- Klassisches Modell:
  - Mehr Platzbedarf:
    - Ein Volume Manager verbindet einzelne Slices oder Platten zu einem RAID



ZFS im Vergleich



- Klassisches Modell - Nachteile:
  - Filesystem weiß nichts von den darunterliegenden Platten
    - Keine Optimierungen wie paralleles lesen und schreiben auf mehreren Platten
    - Spiegel für FS nicht lesbar
  - Volume weiß nichts vom Dateisystem
    - Keine Optimierung eines Schreibzugriffs möglich
  - grow/shrink zeitaufwendig wenn überhaupt machbar

ZFS im Vergleich

- Klassisches Modell:
  - Nachteile:
    - Datenkorruption möglich
      - VM kann nur Prüfsummen direkt nach schreiben bilden
    - Hoher Administrationsaufwand
      - Ein Filesystem erfordert:
        - partitionieren
        - Volume erzeugen
        - FS erzeugen
        - vfstab eintragen
        - dfstab/exporte eintragen

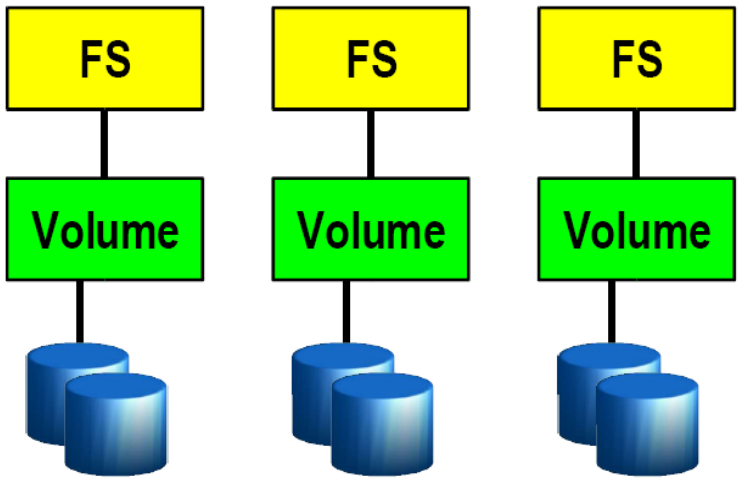
ZFS im Vergleich

- ZFS Modell:
  - Alle Dateisysteme liegen auf einem Pool von Platten.
  - Dateisystem und Pool „wissen“ voneinander
    - Datenintegrität durch „self healing“
    - Optimierungen

FS/Volume Modell vs. ZFS

Traditionelle Volumes

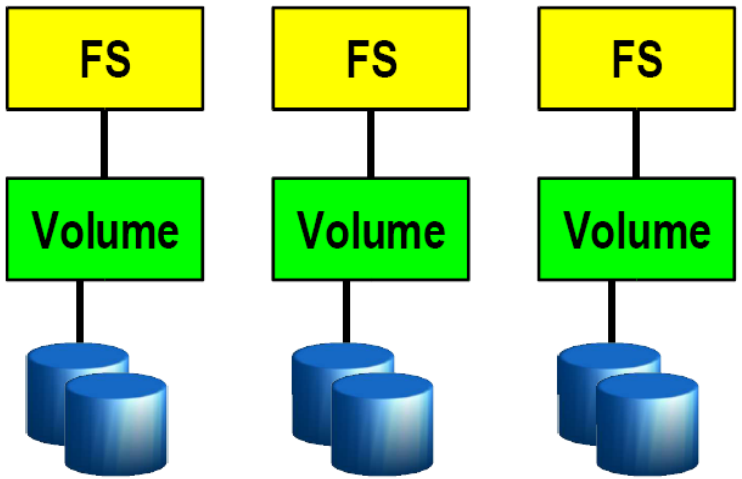
- Abstraktion: virtuelle Disk (fest)
- Volume für jedes Filesystem
- Grow/shrink nur koordiniert
- Bandbreite / IOs aufgeteilt
- Fragmentierung des freien Platzes



FS/Volume Modell vs. ZFS

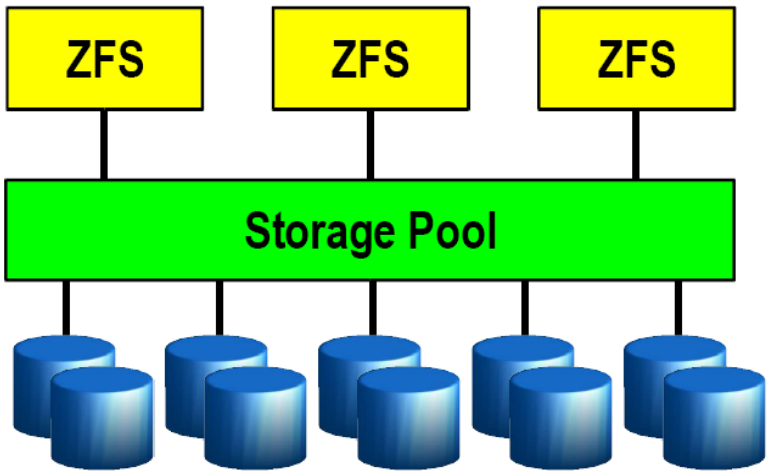
Traditionelle Volumes

- Abstraktion: virtuelle Disk (fest)
- Volume für jedes Filesystem
- Grow/shrink nur koordiniert
- Bandbreite / IOs aufgeteilt
- Fragmentierung des freien Platzes



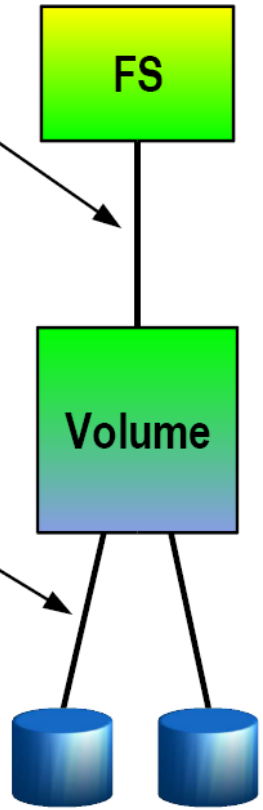
ZFS Pooled Storage

- Abstraktion: Datei (variabel)
- Keine feste Platzeinteilung
- Grow/shrink via Schreiben/Löschen
- Volle Bandbreite / IOs verfügbar
- Freier Platz wird geshart



FS/Volume Model vs. ZFS

FS/Volume I/O Stack



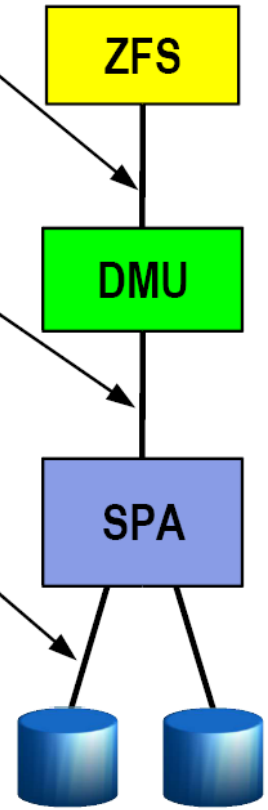
Block Device Interface

- Schreib-Reihenfolge erzeugt quasi-Integrität
- Stromausfall: Metadaten konsistent mittels fsck
- Journaling (nur Metadaten)

Block Device Interface

- Muß so schreiben wie es vom FS kommt
- Stromausfall: ggf. Spiegel ungleich
- Langsam, da synchron

ZFS I/O Stack



Object-basierte Transaktionen

- Daten-Änderungen => Transaktionen
- *All-or-nothing!*

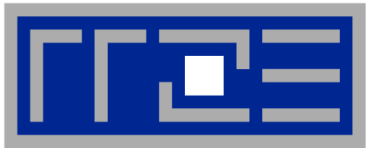
Transaction Group Commit

- Daten-Änderung und zugehörige Metadaten => Transaktions-Gruppen
- Commit mittels COW

Transaction Group Batch I/O

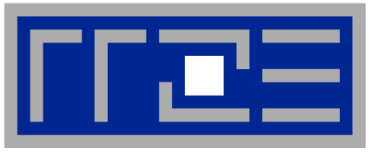
- Zusammenfassung + Optimierung
- Stromausfall -> alter Zustand
- Platten werden ausgenutzt

ZFS Datenintegrität



- Filesystem auf mehreren Platten
- End-to-End Datenintegrität
  - Prüfsumme über jeden Block (Daten und Metadaten)
  - Uberblock (in etwa sowas wie Superblock bei UFS) via Seriennummer (64bit Integer) und SHA256
- Copy-On-Write
  - Daten kopiert, dann verändert, dann original als unbenutzt gekennzeichent

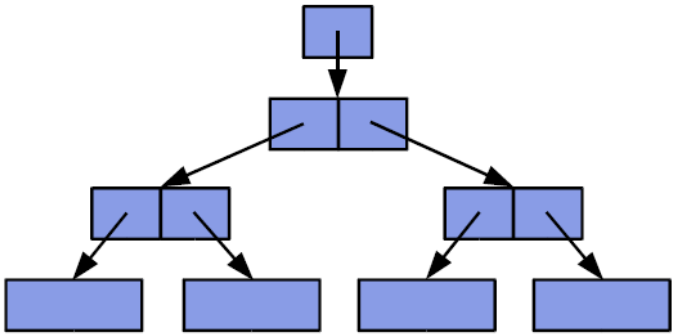
ZFS Datenintegrität



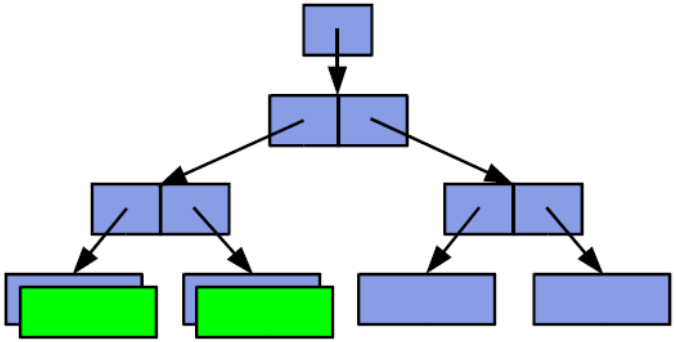
- Transaktionen für alle Operationen
  - -> kein fsck
  - Kein Journaling (Daten immer konsistent)
  - höhere Performance
  - Neue uberblock Version -> neuer Zustand

Copy-On-Write Transaktionen

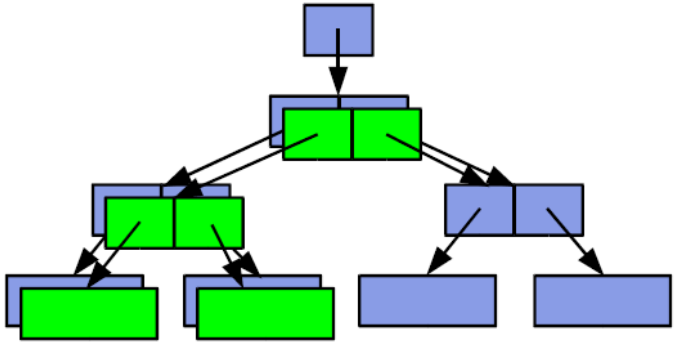
1. Konsistenter Zustand



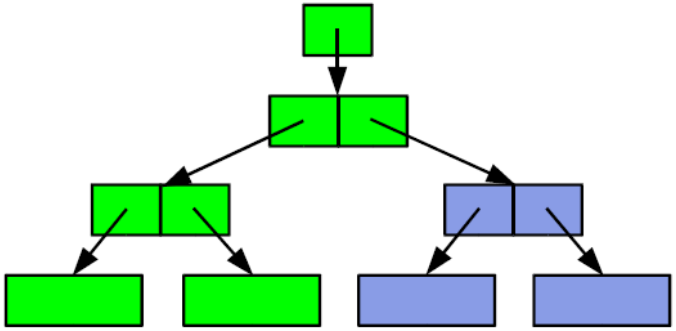
2. Dateien schreiben



3. Zufügen Metadaten



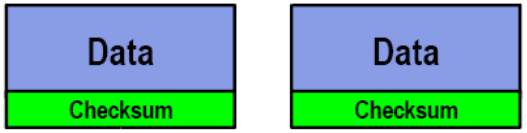
4. Schreiben uberblock (= Commit)



End-to-End Datenintegrität

Disk Block Prüfsummen

- Prüfsummen bei Datenblock
- Auf Disks meist kurz (Fehler unentdeckt)
- Einige Disk Fehler bleiben unentdeckt

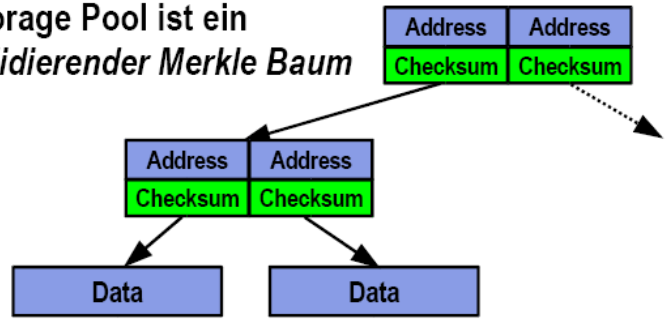


Nur Fehler auf Medium erkennbar

- ✔ Bit rot
- ✗ Phantom writes
- ✗ Misdirected reads and writes
- ✗ DMA parity errors
- ✗ Driver bugs
- ✗ Accidental overwrite

ZFS Daten Integrität

- Prüfsumme bei Adresse
- Gemeinsamer Fehler: unwahrscheinlich
- Storage Pool ist ein *validierender Merkle Baum*



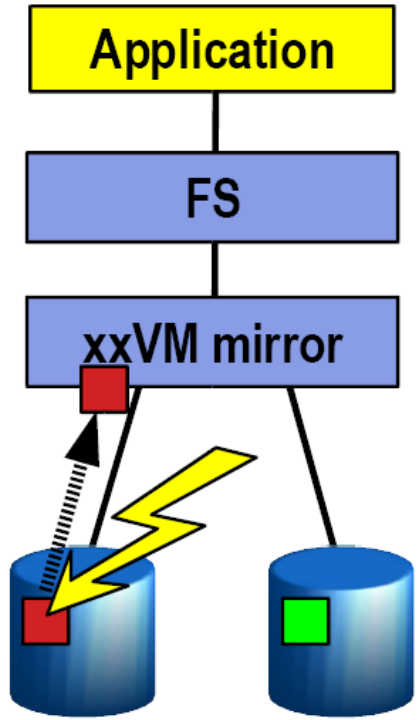
ZFS validiert alle Blöcke

- ✔ Bit rot
- ✔ Phantom writes
- ✔ Misdirected reads and writes
- ✔ DMA parity errors
- ✔ Driver bugs
- ✔ Accidental overwrite

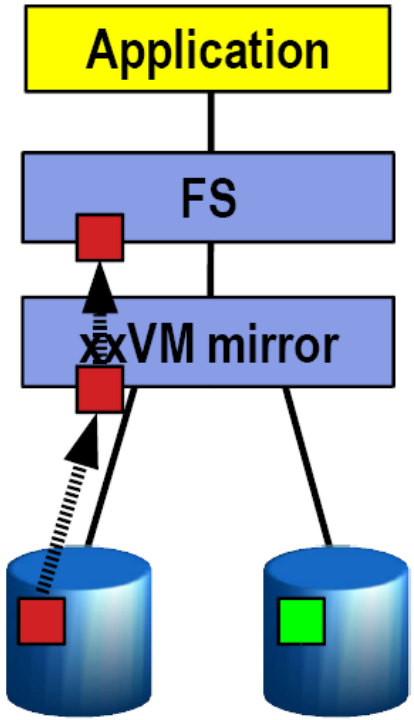
ZFS Datenintegrität

Volume Manager:

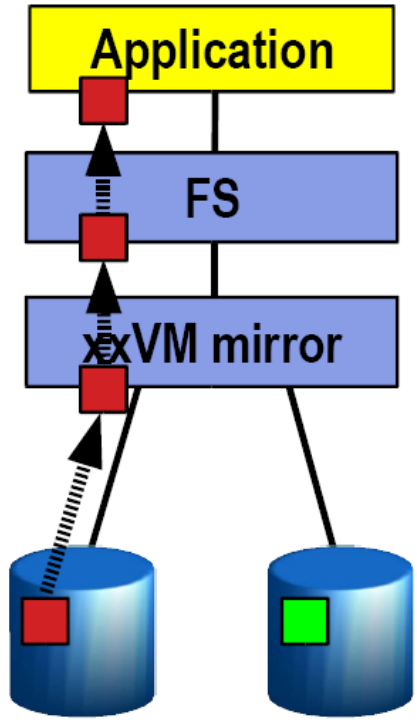
1. read liefert defekten Block



2. Falsche Metadaten:
Filesystem hat Probleme,
Absturz OS möglich



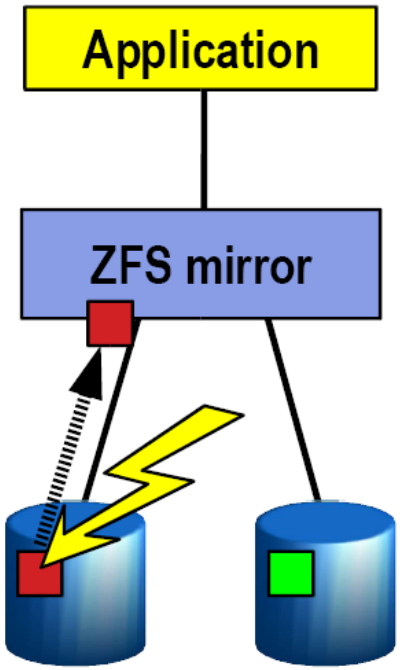
3. Falsche Daten:
Applikation bekommt Probleme
oder rechnet falsch
(ggf. unbemerkt!!!)



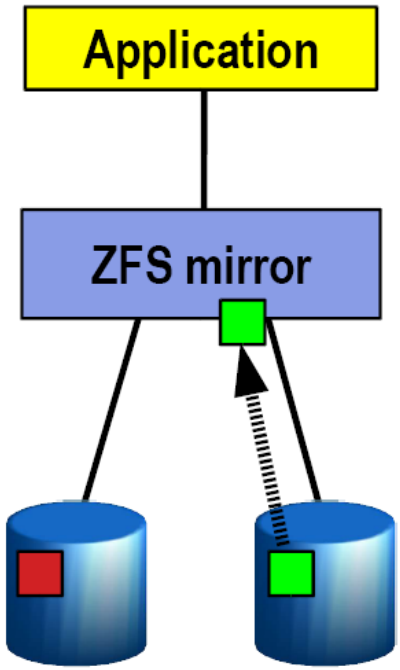
ZFS Datenintegrität

ZFS mit self-healing:

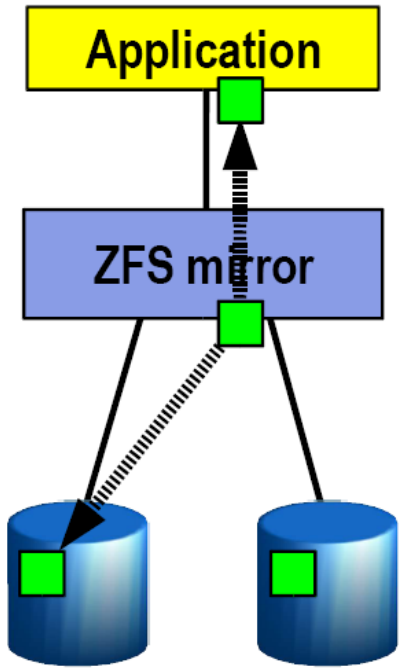
1. read liefert defekten Block



2. ZFS berechnet Prüfsumme; da diese falsch ist, wird der Spiegel gelesen (Metadaten sind also korrekt)



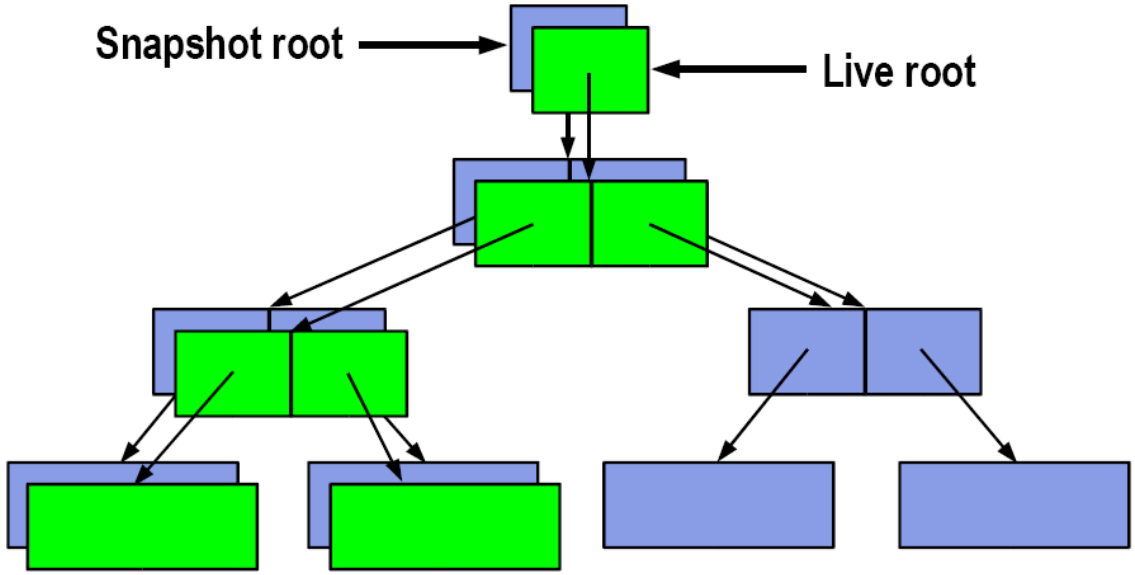
3. ZFS liefert korrekte Daten an die Applikation; UND korrigiert defekten Block!



ZFS Snapshots

Snapshots:

- alte Blöcke werden beibehalten
- Uberblock Informationen werden an andere Stelle kopiert

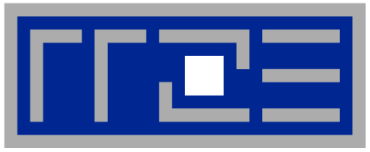


ZFS Clones

Clones:

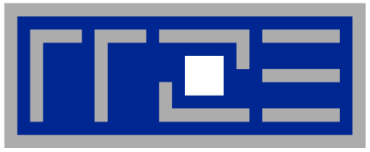
- Einfach gesagt: Snapshots, die beschreibbar sind
- Am Anfang zeigt ein Clone auf dieselben Datenblöcke wie das original, mit den ersten Schreiboperationen divergieren die beiden FS
- Statische Daten (etwa Bibliotheken, OS Dateien) bleiben identisch -> Platzersparnis etwa bei Zonen

ZFS – weitere Besonderheiten



- RAID: Stripe, Mirror und RAID-Z
- Disk Scrubbing: alle Daten regelmäßig gelesen und ggf. korrigiert
- Dynamic striping (Pools können wachsen), alle darauf liegenden Filesysteme wachsen gleichsam mit
- Das erzeugen eines Pools und dann da drauf eines Filesystems geht nicht nur einfach, sondern auch sehr schnell
- Volumes können (endian-unabhängig) zwischen Rechnern transferiert werden

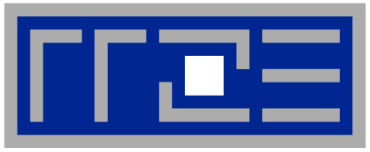
ZFS – weitere Besonderheiten



- ACLs an Windows NTFS ACLs (nicht POSIX) angelehnt und somit sehr feingranular
- Quota, Reservierung
- ZFS „mounted“ sich von selbst
- ZFS „exportiert“ sich von selbst
- Komprimierung der Daten
- (Online Verschlüsselung in Arbeit)

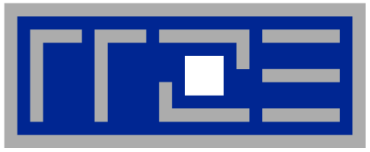
Alles zentral von ZFS aus verwaltet

Verfügbarkeit und Zukunft von ZFS



- ZFS in OpenSolaris bereits verfügbar
- Juni 2006 -> nächsten Solaris 10 Release (Solaris 10 Update 2) sehr wahrscheinlich verfügbar (in einer Beta ist es integriert)
- Online Verschlüsselung der Daten
- Shrinking von Dateisystemen
- Apple will es in OSX
- In DragonFly BSD wird es aktuell integriert

ZFS - Beispiele



- Einfach Administration mit Solaris WebConsole (http://HOSTNAME:6789/zfs)
- oder Kommandozeile:

```
# zpool create testpool c1t0d0 c1t1d0 ...
```

  - Ein Befehl für:
    - Erzeugen eines Pool mit Namen testpool
    - Erzeugt ein FS gleich mit
    - Mounted das FS unter /testpool
    - mount auch nach reboot

ZFS - Beispiele

```
# zpool create mirpool mirror c1t0d0 c1t1d0
# zfs create mirpool/home
# zfs set mountpoint=/home mirpool/home
```

- Erzeugen eines gemirrorten Pool mit Namen mirpool
- Erzeugt eines eigenen ZFS mit Namen home
- mounten des FS unter /home

Beliebige weitere FS:

```
# zfs create mirpool/home/unrz142
# zfs create mirpool/home/unrz06
```

ZFS - Beispiele

 # zpool add mirpool mirror c2t0d0 c2t2d0

- **Vergrößert den Pool um einen weiteren Mirror**

ZFS Properties:

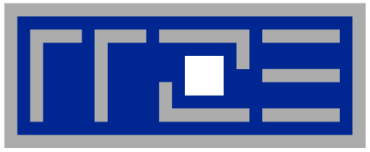
 # zfs set sharenfs=rw mirpool/home

 # zfs set compression=on mirpool

 # zfs set quota=5g mirpool/home/unrz142

 # zfs set reservation=20g mirpool/home/unrz06

ZFS - Beispiele



Snapshots erzeugen:

 # zfs snapshot
mirpool/home/unrz142@Dienstag

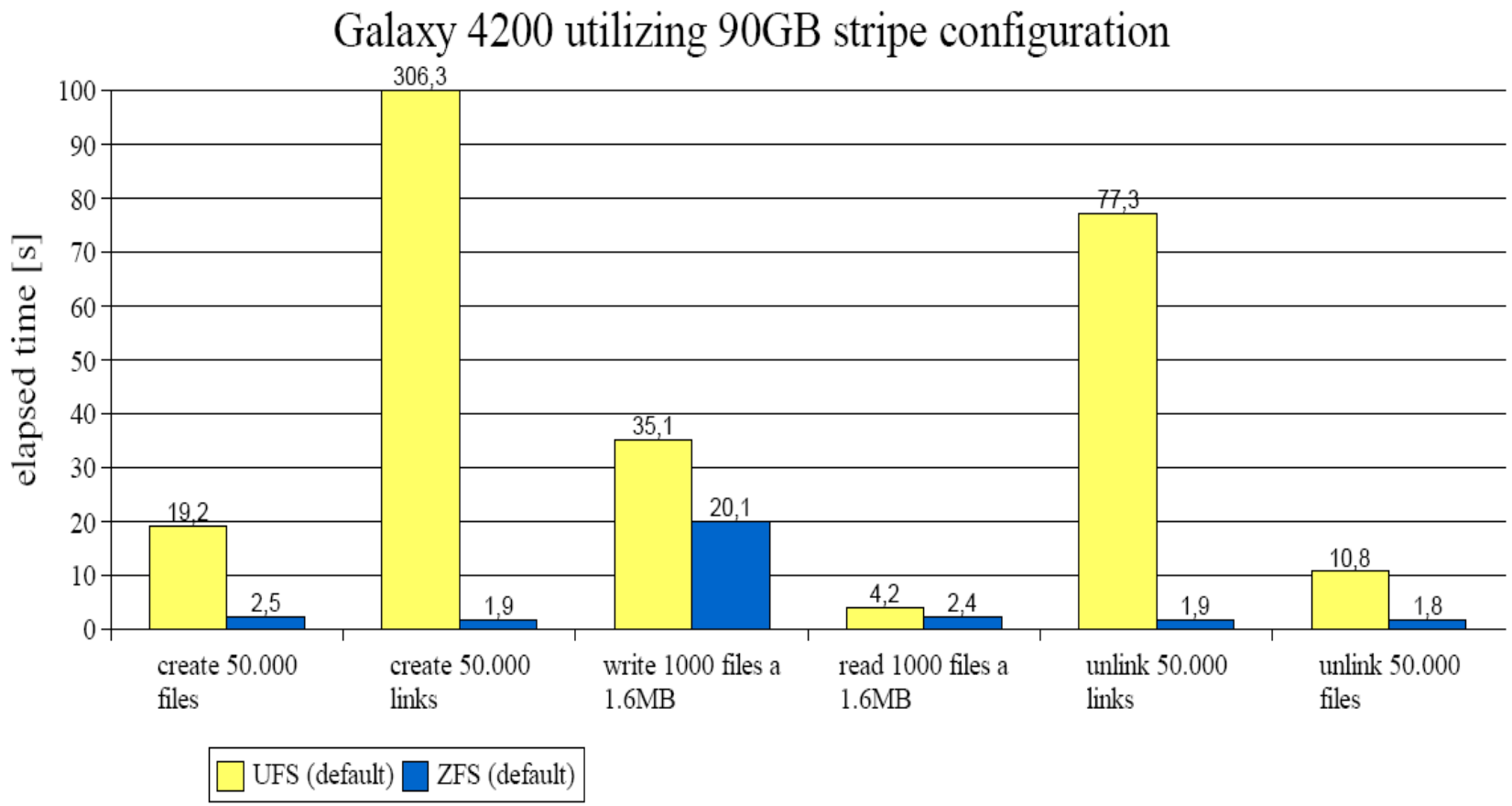
Snapshot rollback:

 # zfs rollback mirpool/home/unrz142@Montag

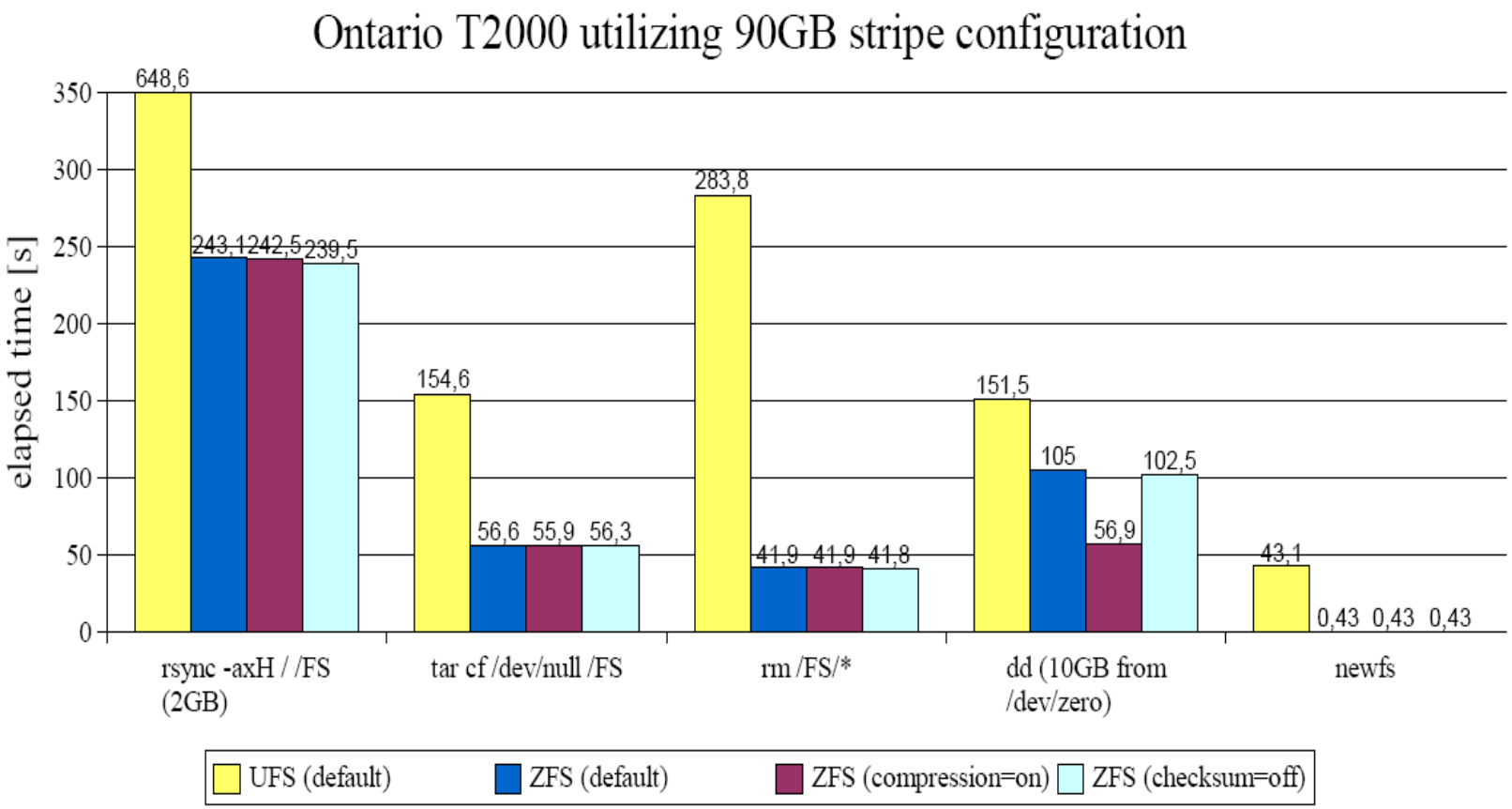
RO Zugriff auf Snaphots:

 # ls ~/unrz142/.zfs/snapshot/Montag/

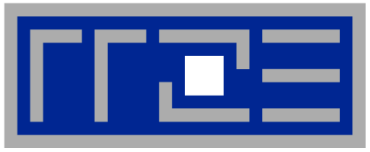
ZFS – Performance von Uni Ulm



ZFS – Performance von Uni Ulm



Abschluß



Danke für Ihre Aufmerksamkeit!

Noch Fragen? Gerne!

Kontakt:
gregor.longariva@rrze.uni-erlangen.de