ORDIX AG

ZFS - Das Dateisystem der nächsten Generation?

Secure Linux
Administration Conference
12.12.2008 Magdeburg

Markus Schreier
ORDIX AG, Wiesbaden
ms@ordix.de
www.ordix.de

ZFS - Das Dateisystem der nächsten Generation?

ORDIX AG

Agenda

ORDIX AG

- Überblick
- Ziel: Größenbegrenzungen aufheben
- Ziel: Datenintegrität
- Ziel: Administrierbarkeit
- Ziel: neue Funktionalität
- Ziel: Geschwindigkeit
- Implementierung unter Linux
- Fazit

Was ist ZFS?

ORDIX AG

- Ein Dateisystem
- Ein Logical Volume Manager
- Von SUN entwickelt
- Frei unter der CDDL
  (Common Development and Distribution License)
- Verfügbar auf
  - Solaris 10
  - FreeBSD
  - MacOS X [1)]
  - Linux [2)]

1) nur beta 2) mit FUSE

Eigenschaften von ZFS

ORDIX AG

ZFS ist keine Erweiterung, sondern eine völlige Neuentwicklung

- Erstes 128-Bit-Filesystem
- Storage Pool basierend
- Unterstützt Kompression, Quota, Platzreservierung, ACLs, NFSv4
- Gewährleistung der Konsistenz („copy on write“)
- Transaktions-orientiert - immer konsistent
- Einfache Administration (...)
- GUI (Java Web Console)

Linux Filesysteme

ORDIX AG

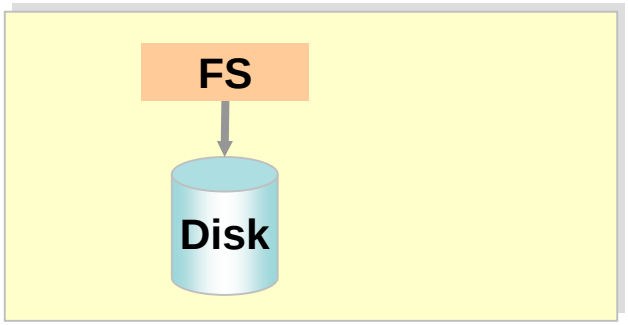
Name	verfügbar seit	Hersteller/Anbieter	Typ
ext2	1993	Linux	herkömmlich
ext3	1999	Linux	journaling
ext4	2006	Linux	journaling
reiserFS	2001	Linux	journaling
OCFS /OCFS2	2002	Oracle Linux	cluster-FS
XFS	1994	SGI irix	journaling
JFS	1991	IBM aix	journaling
VxFS	1991	Veritas	journaling
btrfs	Entwicklungsbeginn 2008	Oracle Linux	COW, 64bit

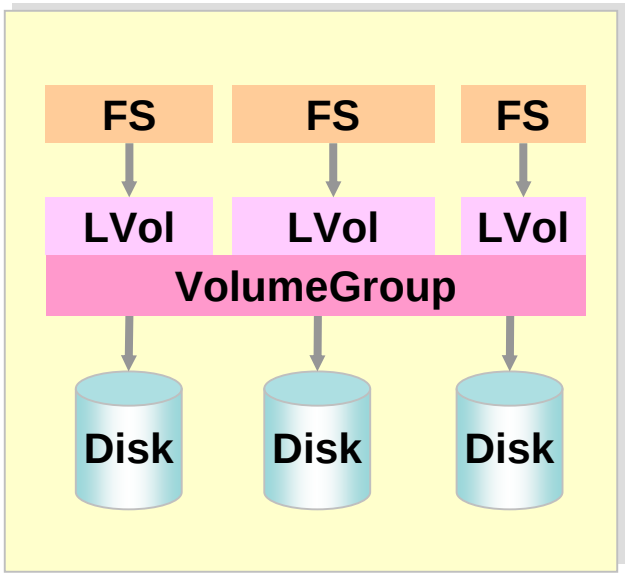
btrfs ist die einzige ernstzunehmende Konkurenz für zfs, jedoch noch in einem sehr frühen Entwicklungsstadium

Dateisysteme – Historie

ORDIX AG

- Dateisysteme (z. B. ext2)
  direkt auf Platten/Devices
- Hinzu kamen neue Ziele:
  - Inkonsistenzen nach System-Crash
    Æ Journaling
  - Mehr Plattenplatz
    Æ striping, concatenating
  - Mehr Sicherheit
    Æ mirror, raid5
  - Mehr Flexibilität
    Æ Softpartitions, LVM, resizefs
  - Mehr Komfort
    Æ LVM (Snapshots, Clone, Komprimierung)



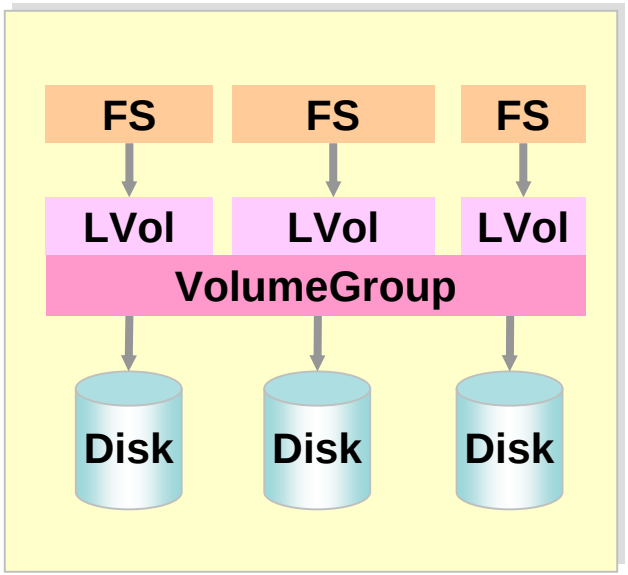


Dateisysteme – Historie

ORDIX AG

Nachteile:

- journal kuriert nur das Symptom
- Filesystem kennt Volume-Struktur nicht
- IO-Sequenzen müssen Reihenfolge beibehalten
- Administration schwierig, mehrstufig (fdisk, lvm, fs)
- Keine Erkennung von korrupten Daten



Ziele von ZFS

ORDIX AG

- Größenbegrenzungen aufheben
  - erstes 128-Bit-Dateisystem
- Datenintegrität
  - erkennt “silent data corruption”
  - Transaktions-orientiert
  - immer konsistent
  - selbstheilend
- Administrierbarkeit
  - integrierter Logical Volume Manager
- Neue Funktionalität
  - Snapshots, Clone
  - ACLs, xAttr, NFS, ...
- Geschwindigkeit

Wie erreicht?

- Ursache der bisherigen Probleme erkannt
- Historie über Board geworfen
- Von bestehenden Annahmen gelöst
- Komplette Neuentwicklung

Agenda

ORDIX AG

- Überblick
- Ziel: Größenbegrenzungen aufheben
- Ziel: Datenintegrität
- Ziel: Administrierbarkeit
- Ziel: neue Funktionalität
- Ziel: Geschwindigkeit
- Implementierung unter Linux
- Fazit

Neue Größenordnungen

ORDIX AG

- In 10 - 15 Jahren reicht 64-Bit-Adressraum nicht aus
  - Æ 128-Bit-FS
  - Æ maximale Größe: 16 ExaByte
- Dynamische Allokation von Metadaten
  - Æ skalierbar

Wert	10er Potenz	Präfix
2^0 = 1	-	
2^10 = 1.024	~10^3	k (kilo)
2^20 = 1.048.576	~10^6	M (Mega)
2^30 = 1.073.741.824	~10^9	G (Giga)
2^40 = 1.099.511.627.776	~10^12	T (Tera)
2^50 = 1.125.899.906.842.624	~10^15	P (Peta)
2^60 = 1.152.921.504.606.846.976	~10^18	E (Exa)
2^70 = 1.180.591.620.717.411.303.424	~10^21	Z (Zetta)
2^80 =	~10^24	Y (Yotta)

Agenda

ORDIX AG

- Überblick
- Ziel: Größenbegrenzungen aufheben
- Ziel: Datenintegrität
- Ziel: Administrierbarkeit
- Ziel: neue Funktionalität
- Ziel: Geschwindigkeit
- Implementierung unter Linux
- Fazit

Ziele bzgl. Datenintegrität

ORDIX AG

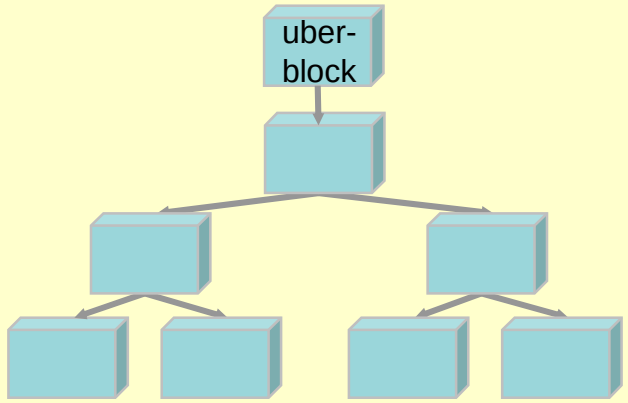
Datenintegrität

- erkennt “silent data corruption”
- Transaktions-orientiert
- immer konsistent
- selbstheilend

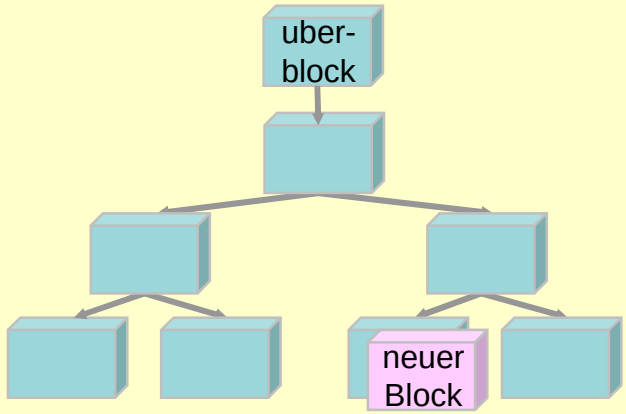
Konsistenz durch "copy on write"

ORDIX AG

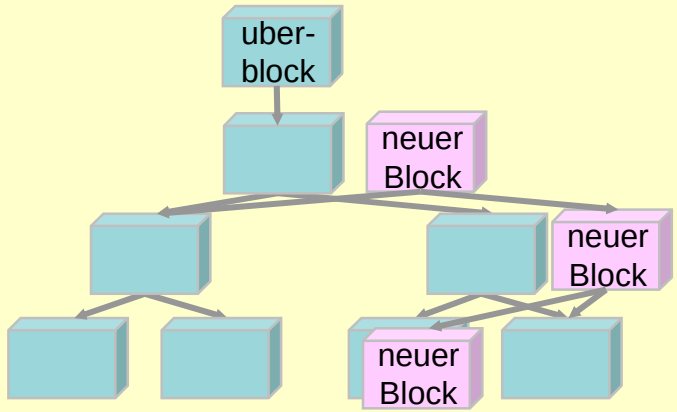
1 Ursprünglicher Block-Baum



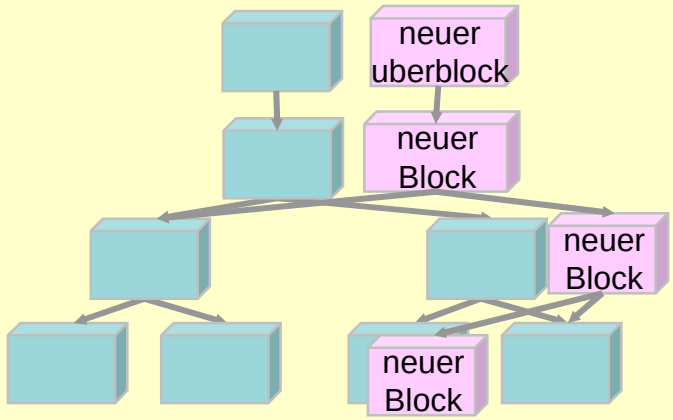
2 copy on write eines Datenblocks



3 copy on write aller referenzierenden Blöcke



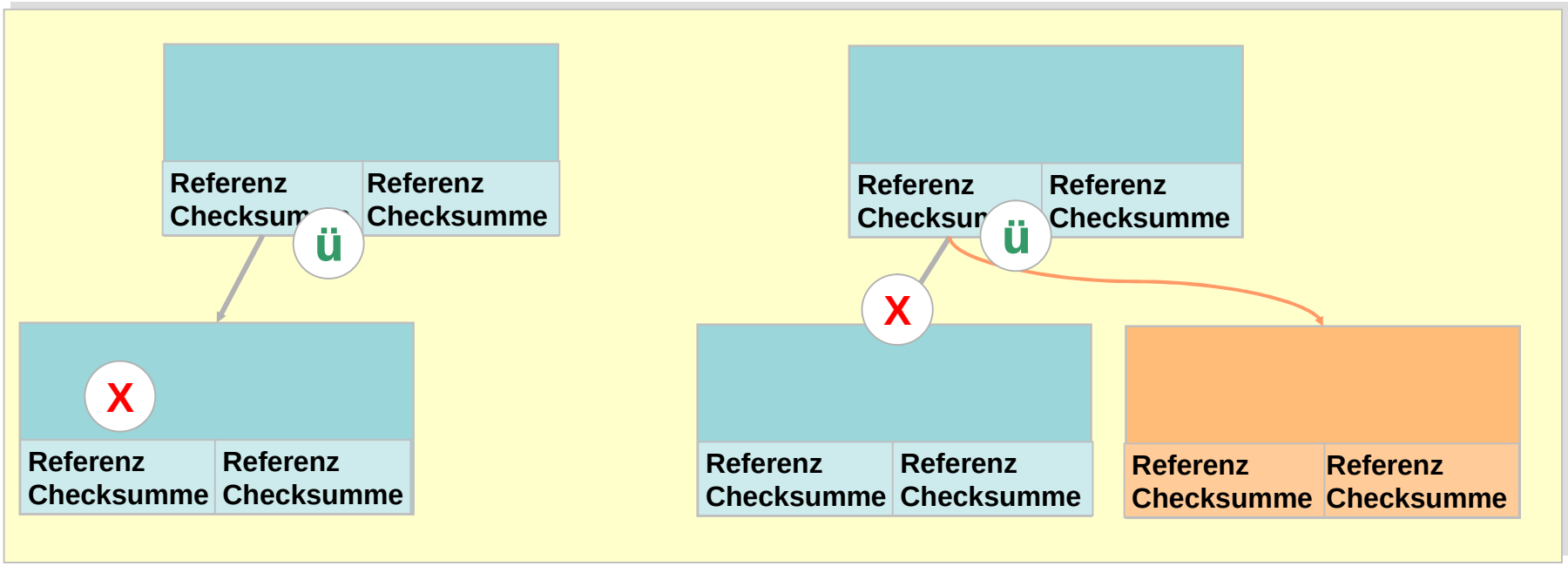
4 copy on write des uberblocks (atomar)



Konsistenz - Checksummen

ORDIX AG

- Ende zu Ende Konsistenz
- Checksummen im referenzierenden Block
  - Æ Fehlerhafte Blöcke werden erkannt
  - Æ Falsche (aber konsistente) Blöcke werden erkannt

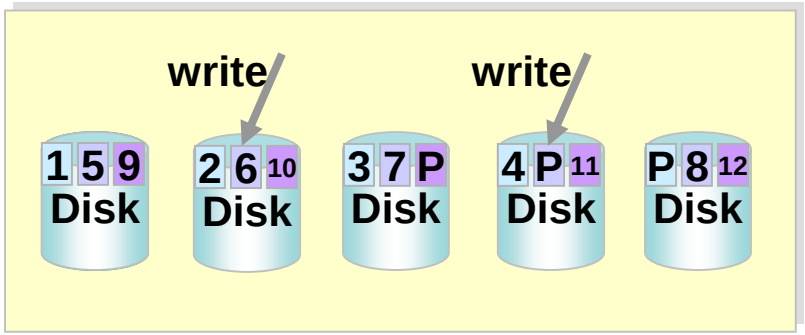


Konsistenz – RAIDZ / RAIDZ2

ORDIX AG

RAID 5 / RAID 6

- Ändern eines Datenblocks
- Ändern von Parity
  - Æ „RAID5 write hole“

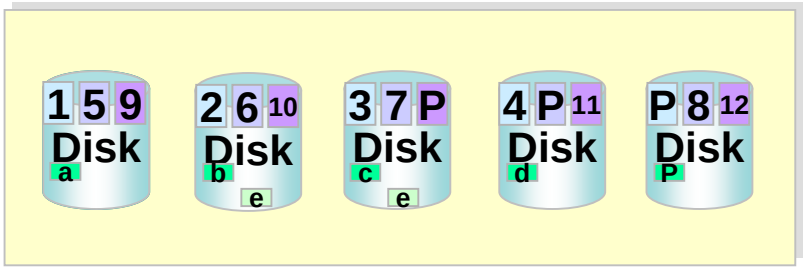


RAIDZ

- Nutzt copy on write
- Schreiboperationen immer über alle Platten
  - Extendgröße wird angepasst
  - sehr kleine Datenmengen als Spiegel
  - Æ Keine RAID5 write hole

RAIDZ2

- Wie RAIDZ, doppelte Parity



Agenda

ORDIX AG

- Überblick
- Ziel: Größenbegrenzungen aufheben
- Ziel: Datenintegrität
- Ziel: Administrierbarkeit
- Ziel: neue Funktionalität
- Ziel: Geschwindigkeit
- Implementierung unter Linux
- Fazit

Administrierbarkeit

ORDIX AG

- fdisk

- pvcreate
- vgcreate
- lvcreate

- mkfs
- fsck
- resizefs
- mount
- tunefs

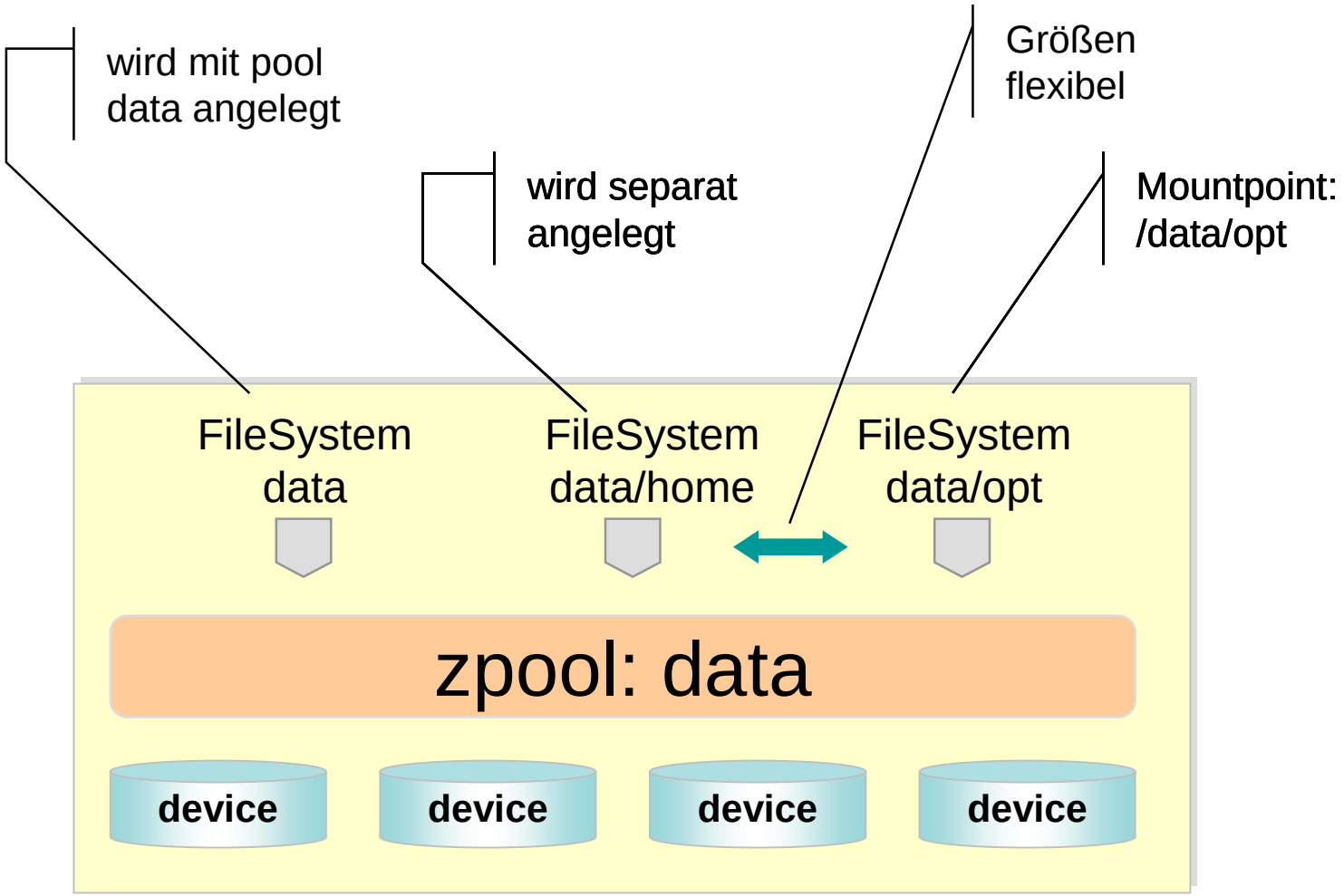
- [un]share
- ...

ext2
ext3
reiserfs
jfs
xfs

- zpool
- zfs

ZFS

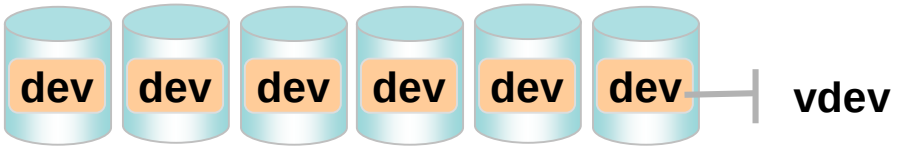
Storage Pools



Poolkonfiguration

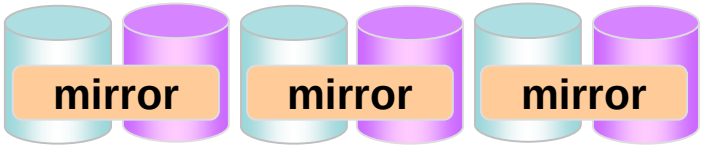
ORDIX AG

Stripe/Concat

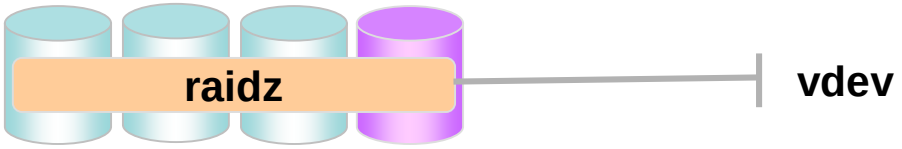


Spiegel

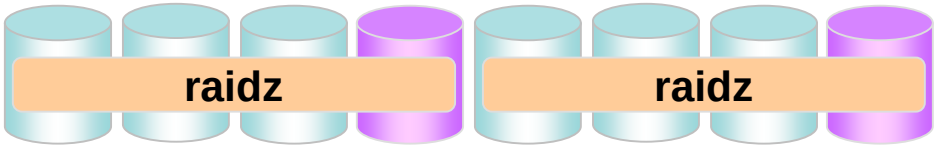
3 Spiegel konkatiniert



RaidZ aus 4 Platten



2 konkatenierte RaidZ



RaidZ2 aus 8 Platten

Pool anlegen

ORDIX AG

zpool create *poolname vdev* [*vdev*[*vdev*[...]]]

vdev kann sein

- einfaches Device
  physdev
- Spiegel Device
  mirror *physdev physdev [physdev [...]]*
- raidz Device
  raidz *physdev physdev physdev [...]*
- raidz2 Device
  raidz2 *physdev physdev physdev [...]*
- Hotspare für redundante Device
  spare *physdev [physdev [...]]*

Bsp.:

```
# zpool create mypool mirror sdb sdc mirror sdd sde
```

Veränderungen am Plattenlayout

ORDIX AG

Ziel `zpool` / `vdev`	mehr Redundanz	mehr Platz	weniger Redundanz	weniger Platz
	`attach dev`	`add vdev`	`dettach dev`	~~`delete`~~
ein Device	Spiegel	stripe/concat	-	-
2er-Spiegel	3er-Spiegel	2x 2er-Spiegel	ein Device	-
Xer-Spiegel	(X+1)er-Spiegel	2x Xer-Spiegel	(X-1)er-Spiegel	-
raidz	-	2x raidz	-	-
raidz2	-	2x raidz2	-	-

- Veränderung am Plattenlayout nur eingeschränkt möglich
- Aber: **Wie oft wird das gebraucht?**

ZFS-Objekte

ORDIX AG

- Pool
- Filesystem
  - poolname/fsname
  - poolname/fsname/fsname
- Volume
  - für Block oder Character-Device
  - Name wie bei Filesystem
- Snapshot
  - gehört zu einem Filesystem
    - filesystem@snapshotname

Beispiele:

- **`mypool/home`**
  Filesystem
- **`mypool/home/ms`**
  Filesystem
- **`mypool/home@Dienstag`**
  Snapshot
- **`mypool@gestern`**
  Snapshot

Automatismen / Administration

ORDIX AG

- Anlegen eines Pools
  - legt erstes Dateisystem implizit an

 # zpool create …

- Anlegen eines Dateisystems
  - erstellt mountpoint
  - mounted das Dateisystem
  - kein Eintrag in /etc/fstab nötig

 # zfs create …

- Steuerung über Dateisystem-Attribute

Reservierung
Quota
Komprimierung
Mountpoint

Beispiel

```
# zfs set reservation=1G mypool/home/ms
# zfs set quota=1G mypool/home/ms
# zfs set compression=on mypool/home/ms
```

Agenda

ORDIX AG

- Überblick
- Ziel: Größenbegrenzungen aufheben
- Ziel: Datenintegrität
- Ziel: Administrierbarkeit
- Ziel: neue Funktionalität
- Ziel: Geschwindigkeit
- Implementierung unter Linux
- Fazit

Snapshots

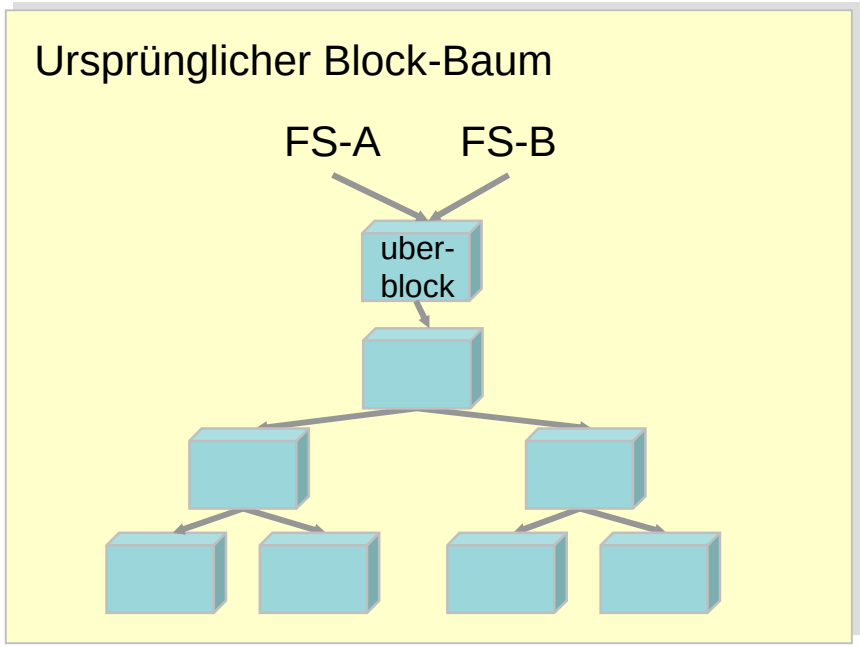
ORDIX AG

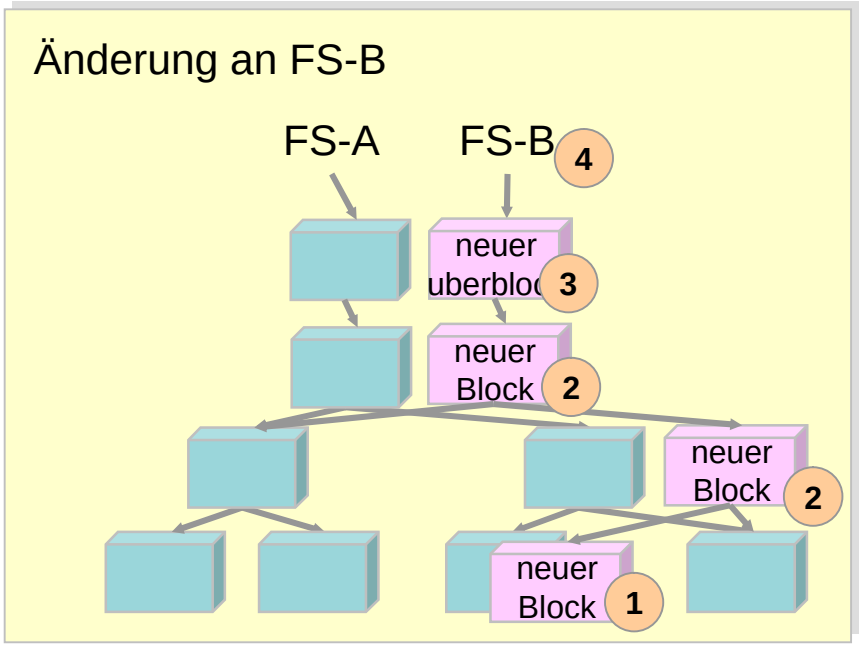
- Keine Performance-Einbußen
- Anlegen von Snapshots:
  - Atomare Aktion rekursiv über mehrere (Unter)-Filesysteme
- Mittel der Datensicherung
  - zfs snapshot -r mypool/home@yesterday
- Lesend Zugreifbar unter [1)]
  - /mypool/home/.zfs/snapshot/yesterday
  - /mypool/home/.zfs/snapshot/2daysago
- Schreibender Zugriff über Clone

1) noch nicht bei Linux

Snapshots / Clone & COW

ORDIX AG





Filesystem A ist Orginal
Filesystem B ist Clone

Bisher keine Änderungen
Alle Filesysteme zeigen
auf gleichen uberblock

1) im FS-B wird ein Block geändert/neu geschrieben
2) darauf referenzierende Blöcke werden neu ge schrieben
3) uberblock wird neu geschrieben
4) Filesystem B verweist auf neuen uberblock

alle nicht veränderten Blöcke bleiben erhalten

Freigaben

ORDIX AG

- NFS-Freigaben [1)]
  - Attribut des ZFS-Filesystems
- SMB-Freigaben [2)]
  - Attribut des ZFS-Filesystems
- iSCSI-Targets [1)]
  - benötigt Volumes (/dev/zvol/….) [1)]
  - Attribut des ZFS-Volumes

1) noch nicht bei Linux
2) noch nicht realisiert

Access Control Lists

ORDIX AG

- Bisher Posix-ACLs
  - setfacl / getfacl
  - ermöglicht UNIX-Rechte weiteren Benutzern/Gruppen zuzuordnen

- NFSv4-ACLs / Windows -ACLs
  - erweitertes chmod
  - zusätzliche Rechte:
    `add_file (w), add_subdirectory (p), delete (d), delete_child (D), execute (x), list_directory (r), read_acl (c), read_attributes (a), read_data (r), read_xattr (R), write_xattr (W), write_data (w), write_acl (C), write_attributes (A), write_owner (o), nicht implementiert: synchronize (s), append_data (p)`

ACL - Vergleich

ORDIX AG

	posix ACL	**Windows, NFSv4 -ACL**
Filesysteme	ext2,reiserfs, nfs (bis v3)	ZFS, NFSv4
Kompatibilität	Posix-draft, Linux, ufs	ähnlich NT-Dateirechten
Einstellen	setfacl	chmod
lesen/ansehen	getfacl	ls -v
Umfang	User-Zuordnung erweitert	Rechte differenziert
Transfer	posix -> nfsv4 - ok	nfsv4 -> posix - nur bedingt

- NFSv4 ACLs sind vielfältiger als Posix ACLs
- Besser für Interoperabilität mit Windows
- ACL = Access Control List
- List bedeutet (hier): mehrere Zeilen mit deny/allow-Regeln

Erweiterte Attribute

ORDIX AG

- Erweiterte Attribute (xAttr)
  - Beliebige Daten/Dateien als Metadaten zu einer Datei ablegen
  - `runat datei [cmd]`
  - wechselt in xattr-Verzeichnis der angegebenen Datei
  - führt dort Kommando cmd aus
  - `cp`, `mv`, `cat`, `ls`, `chmod`, `chown`, ... nutzbar

Agenda

ORDIX AG

- Überblick
- Ziel: Größenbegrenzungen aufheben
- Ziel: Datenintegrität
- Ziel: Administrierbarkeit
- Ziel: neue Funktionalität
- Ziel: Geschwindigkeit
- Implementierung unter Linux
- Fazit

Performance

ORDIX AG

- Transaktionen werden zu Transaktions-Gruppen zusammengefasst
- Intelligente Block-Allozierung (Extents, COW)
- Viele Änderungen ergeben nur eine Schreiboperation
- SUN lieferte viele Performance-Verbesserungen im ZFS-Stack

- Nicht innerhalb einer Platte stripen
- Pool über 80 % belegt => Schreib-Performance-Einbußen
- Ganze Platten verwenden IO-Cache der Platte durch ZFS beeinflussbar

- Besondere Frage bei FUSE

Performance - Vergleich

ORDIX AG

	SLO-Filesystem	COW-Filesystem
Filesystem-Beispiele	statically laid out ext3, reiserfs	copy on write zfs
Random Write in preallocated File	Seek des Blocks schreiben langsam	sequentielles Schreiben schnell
Sequentielles Lesen	Sequentielles IO schnell	Seek des Blocks lesen langsam
Tuning		Große Recordsize => besseres Lesen, schlechteres Schreiben

- Problem für bestimmte Anwendungen
- ZFS in der Entwicklung, Lösungen sind in Arbeit
- Für viele Anwendungen KEIN Problem

ZFS und Datenbanken

ORDIX AG

- Optimierte Datenbanken
  Oracle, Informix, DB/2, Adabas C, ...
  - Optimiert auf Raw Devices (Kopfbewegungen)
  - Æ Daher beste Performance auf Raw Devices
  - Æ Rewrite Filesysteme (UFS, VxFS) gehen auch noch
  - ZFS ist eine zusätzliche Schicht (nochmal Log+Writer), die Performance wegnimmt

- Nicht optimierte Datenbanken
  MySQL, PostGreSQL, MaxDB, SQLite
  - ZFS stellt I/O Optimierung für die Datenbank
  - Æ Beste Performance auf ZFS

Agenda

ORDIX AG

- Überblick
- Ziel: Größenbegrenzungen aufheben
- Ziel: Datenintegrität
- Ziel: Administrierbarkeit
- Ziel: neue Funktionalität
- Ziel: Geschwindigkeit
- Implementierung unter Linux
- Fazit

Jura für Administratoren

ORDIX AG

Linux	ZFS
unterliegt Gnu Public License  GPL **ist** anerkannt von **Open Source Initiative**	unterliegt Common Development and Distribution License  CDDL **ist** anerkannt von **Open Source Initiative**

GPL ist zu CDDL inkompatibel

- Æ ZFS ist nicht in den Linux-Kernel integriert
- Æ ZFS on FUSE

Ausblick:

Jeff Bonwick und
Linus Torvalds

FUSE - Einführung

ORDIX AG

Kernelmodul (hier: fuse.ko)

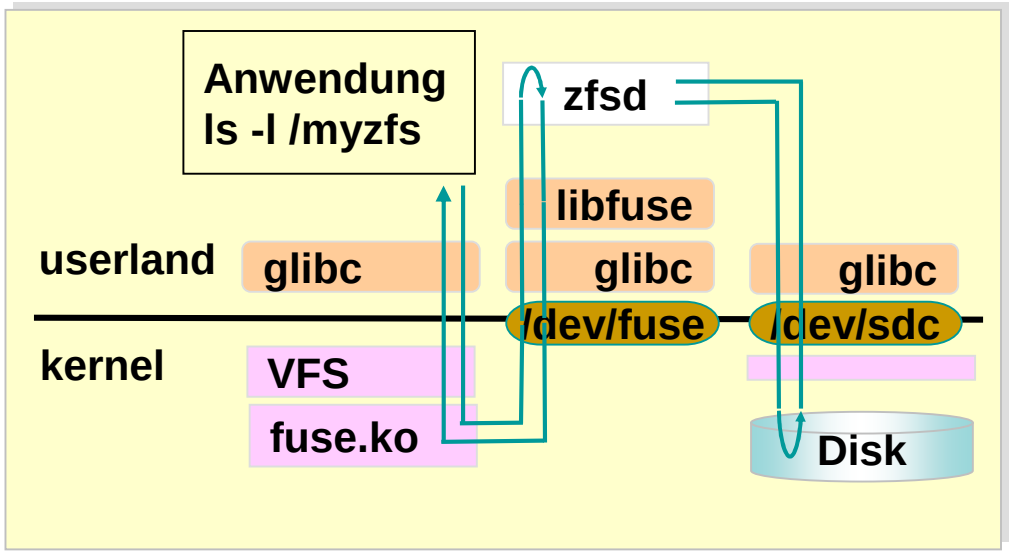
- nutzt Abstraction-Layer VFS (wie ext3, NFS, …)
- übersetzt io-Anfragen
- stellt /dev/fuse bereit
- leitet diese an /dev/fuse weiter

libfuse

- kommuniziert über /dev/fuse
- implementiert calls (open, link, ..)

zfsd

- user-land –Dämon
- nutzt libfuse
- implementiert eigentliches FS
- greift auf Device zu



ZFS und Linux

ORDIX AG

- Benötigt Kernelmodul fuse.ko
- Benötigt Dämon zfs-fuse
- Benötigt `zfs mount –a`

- Kein Zugriff auf Snapshot über `.zfs` –Verzeichnis
- Rollback von zfs-Snapshot macht kein umount/mount
  Æ kann zu IO-Fehlern bei der Anwendung kommen
- promote von Clonen nimmt nicht alle Snapshots mit
- `zfs set mountpoint=`
  - mounted Filesystem nicht wieder automatisch
  - nimmt bestehende Sub-Filesysteme nicht mit

- Volumes noch nicht implementiert

ZFS und Linux - II

ORDIX AG

sharenfs	spielt mit NFS-Server nicht zusammen
sharesmb	bisher nicht lauffähig (auch bei Solaris)
shareiscsi	benötigt Volumes
acl	statt chmod wird zfs allow/unallow verwendet
xattr	benötigt runat
zfs promote	liefert leicht anderes Ergebnis
FS-Hierarchie	Änderungen an übergeordneten Dateisystem greifen nicht durch

Agenda

ORDIX AG

- Überblick
- Ziel: Größenbegrenzungen aufheben
- Ziel: Datenintegrität
- Ziel: Administrierbarkeit
- Ziel: neue Funktionalität
- Ziel: Geschwindigkeit
- Implementierung unter Linux
- Fazit

Fazit

ORDIX AG

- YES: - ZFS ist das Dateisystem der nächsten Generation ,denn
  - ZFS Design & Funktionalität sucht seinesgleichen
- NO: ZFS ist in dieser Generation noch nicht produktiv einsetzbar, denn
  - FUSE ist keine produktiv einsetzbare Lösung für ZFS
  - auch unter Solaris
    muss ZFS bzgl. Performance und Funktion geprüft werden
- Maybe: Es wird vergleichbare Technik unter Linux geben, möglich als
  - ZFS
  - btrfs
  - von Ihnen  geschrieben

ORDIX AG

Vielen Dank
für Ihre Aufmerksamkeit!

einfach.gut.beraten.