Wednesday, October 8. 2008

Anmerkungen zum ZFS-Tutorial in der c´t - revisited

Eigentlich wollte ich das Thema "ZFS-Tutorial in der c´t" mit dem letzte Artikel abschliessen, aber meine Anmerkungen haben doch so viele Reaktionen zur Folge gehabt, das ich mich dazu entschlossen habe, mit einem Artikel darauf zu reagieren, um noch mal einige Dinge zu erlaeutern.

Es scheint da ein Missverstaendnis zu geben. Das von der c´t beschriebene Verhalten tritt nicht schon dann auf, wenn einfach eine Platte ausfällt. Die man page zu zpool schreibt dazu bei der beschreibung zu failmode:Controls the system behavior in the event of catastrophic pool failure.Die Betonung liegt hier auf catastrophic pool failure. Dies ist immer gegeben, wenn nicht mehr ausreichend Redundanzen zur Verfügung stehen, um den Pool aufrecht zu erhalten: Bau ich ein RAID ueber 3 Platten, und ziehe eine Festplatte, dann habe ich genuegend Redundanzen um den Betrieb weiter sicherzustellen. Ziehe ich zwei Festplatten, dann habe ich nur noch eine Festplatte, der Pool ist nicht mehr betriebsfähig, der im letzen Text beschriebene failmode wird wirksam. Vor Build 77 und darauf basierender Distributionen hätte uebrigens an der Stelle das System stumpf gepanict. Das war vorher die Defaulteinstellung. Wie ich schon mal schrieb, Panics sind nicht dazu einfach nur Ungemach zu verbreiten, sie sind dazu entwickelt worden, den Zustand der Daten auf der Platte zu schützen.

Wann passiert sowas? Wenn ich beispielweise der Meinung bin, das alle meine Spiegel hinter dem selben Kabel am selben Controller stecken muessen. Oder vielleicht viele Controller habe, aber durch eine wenig zielfuehrende Konfiguration beide Spiegelhälften hinter einem Controller habe. Macht der Controller oder das Kabel Probleme, sind auf einem Schlag weniger Platten da, als zur Betriebsfähigkeit eines RAID nötig sind und zum Schutz der Daten geht der betroffene ZFS-Pool dann in den von der Option failmode angegebenen Zustand.

Das ist uebrigens auch der Grund, warum es eine ausnehmend schlechte Idee ist, aus einem RAID controller eine LUN rauszumappen, und da einfach nur ZFS drauf zuwerfen. Da kann ich gleich UFS auf das device kippen wenn man so einfach einige der wesentliche Vorteile von ZFS aufgibt (Checksummen koennen dann zwar feststellen, das etwas nicht stimmt, es aber mangels Redundanzen nicht korrigieren. Zudem kommt da genau obiges Problem bei rum. Die ganzen Platten womoeglich noch an einem Ghetto-RAID-Controller ohne Cache-Mirroring. Prelude to disaster.

Ich habe das Verhalten mal mit mehreren Files als vdevs nachgestellt (... habe leider mein Reise-Multipack im anderen Koffer ...). Es ist hier anzumerken ,das auch wenn der erste zpool status blockt ein in einem anderen Fenster durchgefuehrter zpool statusdurchaus schluessige Daten liefert:# zpool status

```
  pool: test
 state: FAULTED
status: One or more devices are faulted in response to IO failures.
action: Make sure the affected devices are connected, then run 'zpool clear'.
   see: http://www.sun.com/msg/ZFS-8000-HC
 scrub: scrub completed after 0h0m with 0 errors on Wed Oct  8 19:06:21 2008
config:

        NAME            STATE     READ WRITE CKSUM
        test            FAULTED      0     0     0  experienced I/O failures
          raidz1        UNAVAIL      0     0     0  insufficient replicas
            /root/test1  UNAVAIL     0     0     0  cannot open
            /root/test2  UNAVAIL     0     0     0  cannot open
            /root/test3  ONLINE      0     0     0
```

Alternativ ist hier auch ein Block in die Logfiles der Fault Management Architecture hilfreich:# fmdump -Ve -c "*vdev.open_failed*"

```
TIME                           CLASS
Oct 08 2008 19:16:34.538548765 ereport.fs.zfs.vdev.open_failed
nvlist version: 0
        class = ereport.fs.zfs.vdev.open_failed
        ena = 0x3670c5e7f4b00001
        detector = (embedded nvlist)
        nvlist version: 0
                version = 0x0
```

scheme = zfs
pool = 0x950cc5a239b45bd7
vdev = 0xd17f1aa49e4ae02b
(end detector)

pool = test
pool_guid = 0x950cc5a239b45bd7
pool_context = 0
pool_failmode = wait
vdev_guid = 0xd17f1aa49e4ae02b
vdev_type = file
vdev_path = /root/test1
parent_guid = 0x27919fd695c7d103
parent_type = raidz
prev_state = 0x1
__ttl = 0x1
__tod = 0x48eceaf2 0x20199a1d

Oct 08 2008 19:17:27.703175938 ereport.fs.zfs.vdev.open_failed
nvlist version: 0
class = ereport.fs.zfs.vdev.open_failed
ena = 0x3706742200c00001
detector = (embedded nvlist)
nvlist version: 0
version = 0x0
scheme = zfs
pool = 0x950cc5a239b45bd7
vdev = 0x6e1d53e952637ff4
(end detector)

pool = test
pool_guid = 0x950cc5a239b45bd7
pool_context = 0
pool_failmode = wait
vdev_guid = 0x6e1d53e952637ff4
vdev_type = file
vdev_path = /root/test3
parent_guid = 0x27919fd695c7d103
parent_type = raidz
prev_state = 0x1
__ttl = 0x1
__tod = 0x48eceb27 0x29e99d02

#Wie bekommte ich das nun wieder in Betrieb, ohne zu rebooten? Platten wieder anstecken und der Befehl zpool clear ist danach dein Freund. Steht ebenfalls in der manpage zu failmode=wait.

Posted by Joerg Moellenkamp in German, Solaris at 19:36

Kannst Du noch eine Aussage machen bezüglich Hostbased Mirroring vs. sychron Replizieren auf dem SAN (z.B. für Fail-over zwischen 2 Datacentern).

Ich finde Ersteres gerade im Zusammenhang mit ZFS interessanter.

Die grossen Hersteller wollen einem immer gleich SRDF, Truecopy oder ähnliches andrehen (generiert ja Umsatz
Anonymous on Oct 9 2008, 07:40

1. Availability Suite (also Instant Image et al.) gibt dir synchrone und asynchrone Mirror fuer einen nicht so hohen Preis Wird auch von Sun Cluster Geo-Edition supported.

2. Host-Based mirror kann auf weite Distanzen problematisch werden.

3. Ansonsten: zfs send/receive
Anonymous on Oct 9 2008, 07:49

2. Das Problem existiert aber bei allen synchronen Replikationsmethoden (?)

3. Nur für asynchron...
   Anonymous on Oct 9 2008, 09:46

Auf große Distanzen kann man ja nen separate ZIL und L2ARC dazwischen schmeißen. Werde ich mal testen wie gut das geht wenn Solaris 10/08 raus ist.
   Anonymous on Oct 10 2008, 11:42