ZFS Resilience Testing

Sun Microsystems has a new file system called ZFS, the Zettabyte File System. There are extensive articles on the subject but here are some salient points:

- Can let file systems span multiple hard drives.
- Built-in mirroring in various ways.
- Allows for snapshots to be made without having to provision space ahead of time.
- Repairs data automatically when one device in a mirror gets corrupted.
- Can address more storage space than the Earth is capable of powering.

As I am now setting about using ZFS on my server, I came to ask whether or not ZFS is reliable. Given small corruptions of the file system as stored on disk, will small, large or complete data loss occur?

I used the same technique as the one I used for True Crypt Resilience Testing. OpenSolaris 2009.06 was installed in VirtualBox with one 8GB hard drive for the system pool and two empty 5GB hard drives. Some nomenclature: ZFS file systems are part of pools. One more or storage devices constitute a pool. The operating system itself resides in the *rpool*. I created a new pool appropriately called *testpool*.

```
zpool create testpool c8d0p0
```

I then created a directory */testpool/games* and added 4,6GB of data to it. Checksums were generated and a snapshot was made to allow me to revert the machine to an uncorrupted state.

I use SystemRescueCD for destroying data. This means that the drive OpenSolaris calls `c8d0p0` suddenly gets the name `/dev/sdb`. Never mind that. More important is the whole ”dd” thing below. The syntax of the dd-command is

```
dd
if=(source) of=(destination)
bs=(block size)
seek=(skip this many blocks in in destination file)
count=(number of blocks)
```

If there is now ”seek” argument we start writing from the beginning of the destination file.

Test 1

The first test was made without any corruption, just to test the system.

```
Corruption:
0 out of 762 files which is 0.0%.
0 B out of 4697.0 MB which is 0.0%.
```

Test 2

Next I zeroed out 105 MB of data starting 3GB from the beginning of the drive:

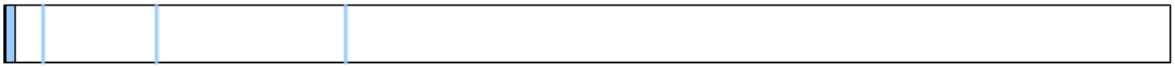
```
dd if=/dev/zero of=/dev/sdb bs=1M seek=3000 count=100
```

Rebooting into OpenSolaris, the check.py-script returned the following information:

```
Corruption:
8 out of 762 files which is 0.0%. (0.01%)
166.0 MB out of 4697.0 MB which is 0.0%. (0.03%)
```

To be honest it only returned 0.0% on both but I added the last digit by using a calculator.

Test 3



So anyway, the data loss was not that much larger than the raw 105MB that I destroyed. I got impatient and first tried a trick gleaned from a report[1] I read about ZFS reliability, erasing the first few megabytes of a ZFS device. That is where important information about the whole drive is stored. There are supposed to be copies at the end of the same device, so hopefully there won't be complete data loss. To make things interesting, three 512KB blocks were zeroed out at an offset of 100MB, 500MB and 1000MB.

```
dd if=/dev/zero of=/dev/sdc bs=1M count=20
dd if=/dev/zero of=/dev/sdc bs=1K seek=100 count=512
dd if=/dev/zero of=/dev/sdc bs=1K seek=500 count=512
dd if=/dev/zero of=/dev/sdc bs=1K seek=1000 count=512
```

Upon reboot the script returned the following information:

```
Corruption:
26 out of 762 files which is 0.03%.
63.0 MB out of 4697.0 MB which is 0.01%.
```

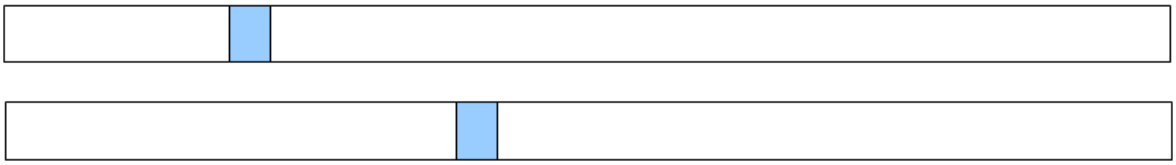
Again I have added the last digit manually. The result is promising. Even though the labels at the beginning of the device were erased the file system could be recovered and mounted without problems(no halted boot sequence or manual override). It might seem like there is a lot of data missing considering that we only erased about 21,5 MB of data. Test 2 yielded 50% more data lost than data directly erased and now we're up to 200% more! Well, my script only checks if the entire file is OK, not every block of data. The data erased was part of a set of files that together contain 63 MB and all those files are more or less corrupted.

Mirroring

While I won't be using ZFS in a mirrored setup on my server(which has only one hard drive), it would be interesting to see how ZFS can repair the type of corruption imposed above when a mirror exists. Thus the other 5GB drive was added to the testpool and the existing hard drive's content was copied over to the new one. ZFS handled that, I only needed to write `zpool attach testpool c8d0p0 c8d1p0.`

A new snapshot was made and a new set of tests were started.

Test 4



The idea here was to erase 100 MB segments of each hard drive, only in different locations. ZFS should have no problem getting all the data back.

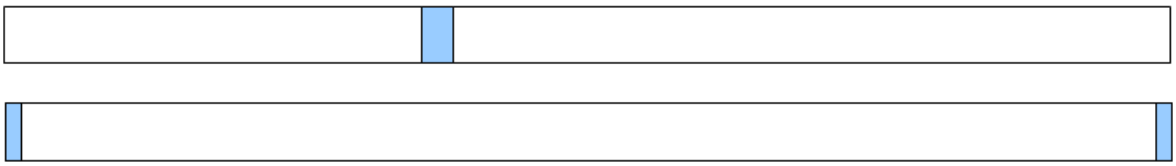
```
dd if=/dev/zero of=/dev/sdb bs=1M seek=1000 count=100
dd if=/dev/zero of=/dev/sdc bs=1M seek=2000 count=100
```

Upon reboot this turned out to be true:

```
Corruption:
0 out of 762 files which is 0.0%.
0 B out of 4697.0 MB which is 0.0%.
```

While I didn't check it manually, had I now shut off the virtual machine, disconnected one of the hard drives in the mirror, I would have got the same result just running against the other drive. This is because ZFS – as mentioned – automatically repairs corrupted data.

Test 5



The previous test was too easy. Let's throw ZFS a curve ball. We now erase the first 50MB and last 50MB of the second drive in the mirror. Then we erase 200MB in the middle of the first drive in the mirror.

```
dd if=/dev/zero of=/dev/sdb bs=1M seek=2000 count=200
dd if=/dev/zero of=/dev/sdc bs=1M count=50
dd if=/dev/zero of=/dev/sdc bs=1M seek=5100 count=50
```

Here is what OpenSolaris tells us when we reboot.

```
     NAME        STATE     READ WRITE CKSUM
     testpool    ONLINE       0     0     0
       mirror    ONLINE       0     0     0
         c8d0p0  ONLINE       0     0     0
         c8d1p0  UNAVAIL      0     0     0  corrupted data
```

So OpenSolaris already knows that c8d1p0 is busted but seems to believe that c8d0p0 is OK. Let's see if it can live up to its boast by running the script:

```
Corruption:
13 out of 762 files which is 0.01%.
483.0 MB out of 4697.0 MB which is 0.10%.
```

Now OpenSolaris acknowledges that there is data loss:

```
     NAME        STATE     READ WRITE CKSUM
     testpool    DEGRADED     0     0     8
       mirror    DEGRADED     0     0    37
         c8d0p0  DEGRADED     0     0    37  too many errors
         c8d1p0  UNAVAIL      0     0     0  corrupted data
```

This shows how ZFS corrects errors as it runs into them. In this case it can't correct the errors because I've completely mangled c8d1p0. If I would have come across this kind of error in a live environment I would have ordered a ”scrub” to make sure the single remaining drive was OK. This way was more demonstrative.

Conclusion

One might think that a file system where deleting a few megabytes at the right places causes the entire volume to become unrecoverable would get my seal of **dis**approval, but no. If those two crucial sections are both erased that is almost certainly going to be as a result of the entire drive being erased, at which point the apparent vulnerability becomes unimportant. I can't say that ZFS is more reliable than ext3 or NTFS when running on a single drive but it is resilient against substantial data corruption even under those circumstances.

[1] http://pages.cs.wisc.edu/~kadav/zfs/zfsrel.pdf