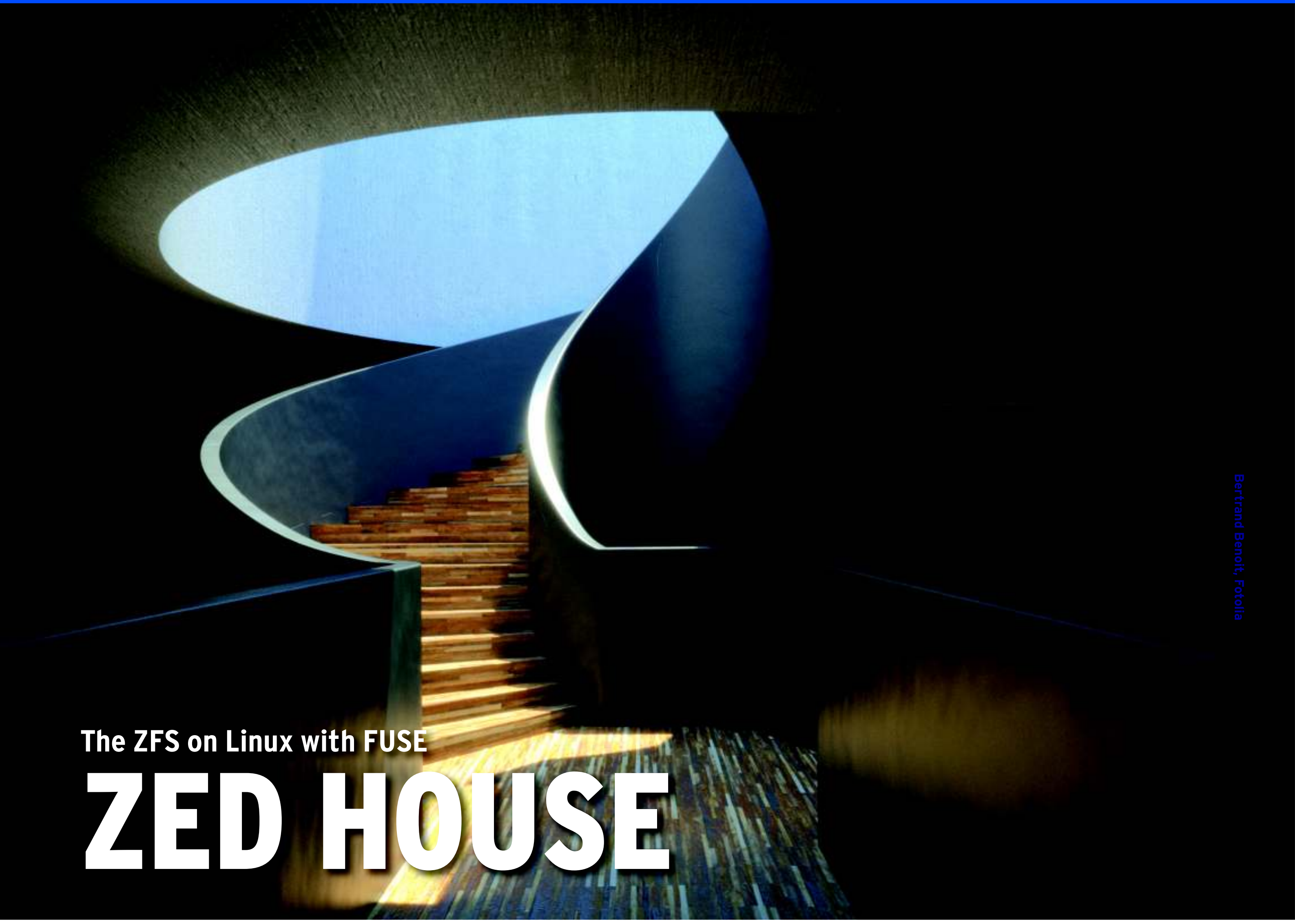

The ZFS on Linux with FUSE

ZED HOUSE

License issues prevent the integration of ZFS with the Linux kernel, but Linux users can try the highly praised filesystem in userspace.

BY CHRISTIAN MEYER

Sun's Zettabyte File System (ZFS) [1] was officially introduced to (Open)Solaris [2] in June 2006, replacing the legacy UFS (Unix File System). ZFS is a 128-bit filesystem with a number of interesting features, such as improved safeguards against defective disks and the ability to manage large numbers of files. Because currently there are no 128-bit data types, ZFS uses the first 64 bits and pads the rest of the structure, ignoring the unused bits in normal operations. The 128-bit design will make it easier to migrate to 128-bit types some time in the future.

The Logical Volume Manager (LVM) lets ZFS pool physical media (drives or partitions). Native RAID functionality allows users with more than two hard disks to set up a RAID pool. (Compared with RAID 5, RAID Z in ZFS has faster write access and is safer if your hardware fails.)

ZFS's list of capabilities includes an automatic snapshot feature to save filesystem states. ZFS only stores the vector to the previous snapshot. This design lets the filesystem create "clones." In contrast to a snapshot, a clone supports read and write access. ZFS also makes it easy to add new hard disks or replace defective disks on the fly. Online compression, which you might remember from NTFS, is another useful extra.

ZFS and Linux

Sun has released ZFS under the free, but non-GPL-compatible CDDL license. License compatibility problems currently prevent the possibility of ZFS integration with the Linux kernel, but no genuine replacement is in sight: right now, ZFS has a good head start on its competitors – except for Oracle's Btrfs [3], which has similar features.

Luckily, ZFS is also available as a FUSE (Filesystem in Userspace [4]) module, which makes it possible to use ZFS on Linux. The current version 0.5 of ZFS FUSE [5] is stable, and it performed well

Installing ZFS

To set up ZFS on Ubuntu, just add the following entry to your repository file /*etc/apt/sources.list*:

```
deb http://ppa.launchpad.net/↩
brcha/ubuntu release_name ↩
main multiverse ↩
restricted universe
```

First, replace *release_name* with *gutsy*, *hardy*, *intrepid*, or *jaunty* as needed to match your release. Then type

```
apt-get update &&
apt-get install zfs-fuse
```

to install the software.

Once you complete this installation, you will be working with the *zfs* and *zpool* commands at the command line.

in our lab. Unlike conventional filesystems that operate in kernel space, FUSE operates in userspace, which means you can expect some performance hits in certain circumstances. If performance is an issue, you might consider moving to Solaris or some BSD variant, in which ZFS is already part of the kernel thanks to the less restrictive BSD license.

Zpools

As I mentioned earlier, ZFS manages individual disks or whole disk arrays as pools. The *zpool* tool is used to create a pool. When creating a pool, it does not matter whether you are working with complete disks, multiple partitions, or, in the simplest case, files. Here, I focus primarily on files, but it is not difficult to apply the concept to hard disks.

For test purposes, Listing 1 creates eight virtual disks for ZFS. Note that ZFS needs at least 64MB per file. The first step is to run *zpool* to create the pool (Listing 1, line 2).

ZFS does not let you reduce the size later, in contrast to XFS. The pool now has a size of 256MB, and you can add new disks to increase the pool size (Listing 1, line 3); also, you can replace individual parts of the pool: The command in line 4 of Listing 1 replaces virtual disk

Mirroring

Mirroring of two disks is the equivalent of RAID Level 1. The system writes data to both disks, providing full redundancy. The failure of one disk does not entail data loss. An optional hot spare disk can step in to replace the defective disk in case of a failure.

Listing 1: Virtual Disks

```
01 $ for i in $(seq 8); do dd if=/dev/zero of=/tmp/$i bs=1024 count=65536; done
02 $ zpool create testpool /tmp/1 /tmp/2 /tmp/3 /tmp/4
03 $ zpool add testpool /tmp/5
04 $ zpool replace testpool /tmp/1 /tmp/6
```

Listing 2: RAID-Z

```
$ for i in $(seq 3); do dd if=/dev/zero of=/tmp/rpool$i bs=1024 count=65536; done
$ zpool create rpool raidz /tmp/rpool1 /tmp/rpool2 /tmp/rpool3
```

Listing 3: Replacing the Disks

```
$ for i in $(seq 4 6); do dd if=/dev/zero of=/tmp/rpool$i bs=1024 count=128000;
done
$ zpool replace rpool /tmp/rpool1 /tmp/rpool4
$ zpool replace rpool /tmp/rpool2 /tmp/rpool5
$ zpool replace rpool /tmp/rpool3 /tmp/rpool6
```

Listing 4: Adding Pools

```
$ zpool add rpool mirror /tmp/rpool4 /tmp/rpool5
$ zpool add rpool raidz /tmp/rpool4 /tmp/rpool5 /tmp/rpool6
```

1 with virtual disk 6. In practical conditions, the user will not notice this replacement. However, this variant is ineffective if one of the media has failed: if this failure happens before you complete the replacement, you will lose data.

The *zfs list* command gives you a useful overview, including the pool name, the disk space used, and the mount point. The *zpool iostat -v* command gives you details of read and write operations.

Adding Fail-Safes

Disk mirroring (aka RAID 1) is a simple approach to adding a fail-safe system (see the box titled "Mirroring"). Another RAID type that protects your data against hard disk failure, RAID 5, requires at least three disks, which is more expenditure for hardware, but with today's hard disk prices, buying three 500GB disks isn't going to cost a fortune. The effective storage capacity is calculated as follows: (number of disks – 1) x (size of the smallest disk). Three 500GB disks give you a total capacity of 1TB.

RAID 5 (single parity) does not lose data if one disk in the array fails. Additionally, you can reconstruct the data from the defective disk on a swap, but if another disk fails before you have finished reconstruction, you lose all the data on the array. In other words, you have to be quick about providing a replacement for the defective disk.

RAID 6 improves redundancy and data protection with the use of double parity: A single disk failure will not faze the system; losing a second disk puts the array in an unsafe state. RAID 6 needs at least four disks and is thus fairly expensive because you lose two disks for parity data storage.

Just like the software-based RAID on Linux, ZFS RAID-Z and RAID-Z2 work similarly to RAID 5 and RAID 6, respectively. However, ZFS handles all write operations in a way that transfers the data and checksums atomically to ensure consistent data in case of a power failure. One big advantage is that you do not need an expensive hardware RAID controller. Single- or dual-core CPUs cost a fraction of what a controller costs and are fast enough to handle the RAID controller's tasks.

Listing 5: zlist Output

```
$ zfs list
NAME           USED  AVAIL  REFER
MOUNTPOINT
rpool          409K  266M  32,2K  /
rpool
rpool@created    0     -   32,2K  -
```

Using RAID

Listing 2 demonstrates a RAID-Z array with three (virtual) disks – RAID-Z2 is similar. (The keyword for RAID-Z2 is *raidz2* instead of *raidz*.) The commands in Listing 2 create a pool with RAID functionality. Note that ZFS will not let you extend the capacity: You can't just add new disks to the RAID pool. However, there is a workaround. As shown in Listing 3, you can replace the existing disks with three larger disks.

Alternatively, you can increase the array capacity by adding mirror, or RAID-Z, pools to the existing pool, rpool (Listing 4). This technique makes sense when the new disks are the same size as the existing disks. As long as you have more than two disks, RAID-Z is preferable to mirroring for failure safety reasons.

Preventing Data Loss

Modern hard disks have self-test functions that let you check current hardware status by running a special tool. If a disk is in a critical state, ZFS lets you remove it from the pool to check the hardware:

```
zpool offline rpool /tmp/rpool3
```

If you find out the hardware has an irreparable defect, you have no alternative but to replace it with the use of the *zpool replace* command, as shown in Listing 3. Whereas *offline* simply disables the disk in the array, the *replace* command swaps the existing medium.

ZFS then proceeds to synchronize the pool, which can take a couple of minutes. The *zpool status* command keeps you up to date with the current status.

If you are wondering why Apple is so interested in ZFS, you might consider an interesting feature in Mac OS X: The "Time Machine" stores filesystem states, which lets users restore older states. Time Machine is actually based on ZFS.

In OpenSolaris, the developers have integrated this feature with Nautilus [6]. On Linux, you currently have no alternative but to use the command line. To create a snapshot, type *zfs snapshot rpool@created*. The @ sign and an arbitrary string following it are important. The *zfs list* command outputs the existing pools and snapshots (Listing 5).

If you change a pool – that is, copy or add files – the *USED* and *REFER* columns will have changed from the original time. If you accidentally delete some data, *zfs rollback rpool@created* is all it takes to restore the pool to its original state.

Conclusions

Compared with the current crop of popular Linux filesystems, ZFS has some very interesting features, such as the integration of the Volume Manager and RAID and the ability to create snapshots. Other promising traits include online compression, or the ability to export or import pools. The many benefits of ZFS make it quite clear how big a lead this filesystem has over its competitors right now. Although Oracle's Btrfs promises similar abilities, it will take some time until it is ready for production use. ■

INFO

[1] ZFS: *http://opensolaris.org/os/community/zfs/*

[2] OpenSolaris: *http://www.opensolaris.com*

[3] Btrfs: *http://btrfs.wiki.kernel.org/index.php/Main_Page*

[4] FUSE: *http://fuse.sourceforge.net/*

[5] ZFS FUSE: *https://developer.berlios.de/projects/zfs-fuse/*

[6] Snapshot integration in Nautilus: *http://blogs.sun.com/erwann/entry/zfs_on_the_desktop_zfs*

THE AUTHOR

Christian Meyer has worked with Linux since the mid-1990s and was a member of the Gnome Deutschland e.V. board between 2004 and 2006. In his leisure time, Christian enjoys playing badminton or researching new technologies for Linux and Solaris.