THE UNIVERSITY of EDINBURGH

GridPP UK Computing for Particle Physics

22nd International Conference on Computing in High Energy and Nuclear Physics

# Evaluation of ZFS as an efficient WLCG storage backend based on ZFS ON LINUX

Marcus Ebert
University of Edinburgh

- ZFS and ZFS on Linux
- Hardware raid vs ZFS software raid
- ZFS storage setup at Edinburgh
- Compression effects

## ZFS on Linux

- ZFS was developed by SUN Microsystems for Solaris
  - became OpenSource together with OpenSolaris in 2005
  - license: CDDL, which is unfortunately not compatible with GPL
    - reason why it is not in kernel source and why ext4/xfs still used during the last 10 years
- on Linux, early tries to implement it through fuse
  - poor performance
- later native kernel modules became available
  - distribution of source code that one has to compile
  - DKMS usable
  - first stable version back in 2013
- storage solutions based on it at enterprise-level exist
  - JovianDSS by Open-E, TrueNAS by iXsystems
- available in repositories for many Linux distributions
  - repositories for RH/SL/CentOS exists
  - fully integrated in Ubuntu 16.04, despite the license problem...

# ZFS on Linux

- ZFS was developed by SUN Microsystems for Solaris
  - became OpenSource together with OpenSolaris in 2005
  - license: CDDL, which is unfortunately not compatible with GPL
    - reason why it is not in kernel source and why ext4/xfs still used during the last 10 years
- on Linux, early tries to implement it through fuse
  - poor performance
- later **native kernel modules became available**
  - distribution of source code that one has to compile
  - **DKMS usable**
  - first stable version back in 2013
- **Storage solutions based on it at enterprise-level exist**
  - JovianDSS by Open-E, TrueNAS by iXsystems
- available in repositories for many Linux distributions
  - **repositories for RH/SL/CentOS exists**
  - **fully integrated in Ubuntu 16.04, despite the license problem...**

## What is ZFS

a very small selection of features...

- pooled storage system that combines file system, volume manager, and raid system
- 128 bit transactional file system
- COW file system that transforms random writes into sequential writes
- protection against silent data corruption and has self healing of data
- supports on-disk compression
- stripe size and block size are both variable
- moves data and disk management out of the OS into the file system
- simplifies storage management and has very easy administration

# What is ZFS

a very small selection of features...

- **pooled storage system that combines file system, volume manager, and raid system**
- 128 bit transactional file system
- COW file system that transforms random writes into sequential writes
- **protection against silent data corruption and has self healing of data**
- **supports on-disk compression**
- stripe size and block size are both variable
- moves data and disk management out of the OS into the file system
- **simplifies storage management and has very easy administration**

## Storage setup

## Traditional GridPP storage system @Edinburgh

- locally distributed storage
  - DPM headnode
  - many DPM clients that have data storage space available

- mix of DPM client machines exist:
  - 12GB RAM, 8/16C E5620 2.4GHz
    single hardware raid controller, 36x2TB NL6 7.2k, in DAS (3 external enclosures)
    RAID6, 17 disks + 1 HS (2x)
  - 24GB RAM, 8/16C E5620 2.4GHz
    2 hardware raid controllers, 36x2TB NL6 7.2k, 12xinternal and 24 in 2 external enclosures)
    RAID6, 11 disks + 1 HS and 23 disks + 1HS
  - 64GB RAM, 12/24C E5-2620v2 2.1GHz
    2 hardware raid controllers, 36x4TB NL6 7.2k, 12xinternal and 24 in 2 external enclosures
    RAID6, 11 disks + 1 HS and 23 disks + 1HS

- disks on different controllers can't be combined
  each machine with 2x(raid6+1HS) resulting in 30 data disks per machine

# Traditional storage server setup

1. creating and initialization of the hardware raid
   - over 11 to 23 disks, 22TB to 92TB
   - takes more than 1 week for 49TB raid60, read/write performance reduced while ongoing (priority changed to 100%)
2. use *fdisk* to partition the virtual disks
   - fast, but additional manual step
   - ext4 is still limited to 16TB partitions in RH6/SL6
3. use *mkfs* to create filesystem
   - mkfs.xfs on 49TB: 4min:30s (after raid initialization finished)
   - mkfs.ext4 on 15TB partition: 11min:52s (after raid initialization finished)
   - mkfs.ext4 on single 8TB partition can take more than 1h during raid initialization
   - can take hours or even whole day just for formatting whole disk space on a server
4. *mkdir* for mount point
5. edit */etc/fstab*
6. *mount*

# Traditional storage server setup

1. creating and initialization of the hardware raid
   - over 11 to 23 disks, 22TB to 92TB
   - takes more than 1 week for 49TB raid60, read/write performance reduced while ongoing (priority changed to 100%)
2. use *fdisk* to partition the virtual disks
   - fast, but additional manual step
   - ext4 is still limited to 16TB partitions in RH6/SL6
3. use *mkfs* to create filesystem
   - mkfs.xfs on 49TB: 4min:30s (after raid initialization finished)
   - mkfs.ext4 on 15TB partition: 11min:52s (after raid initialization finished)
   - mkfs.ext4 on single 8TB partition can take more than 1h during raid initialization
   - can take hours or even whole day just for formatting whole disk space on a server
4. *mkdir* for mount point
5. edit */etc/fstab*
6. *mount*

repeat steps 4-6 if new OS gets installed.

Make sure the mount point is always the same or files will not be usable by DPM.

# Traditional storage server setup

1. creating and initialization of the hardware raid
   - over 11 to 23 disks, 22TB to 92TB
   - takes more than 1 week for 49TB raid60, read/write performance reduced while ongoing (priority changed to 100%)
2. use *fdisk* to partition the virtual disks
   - fast, but additional manual step
   - ext4 is still limited to 16TB partitions in RH6/SL6
3. use *mkfs* to create filesystem
   - mkfs.xfs on 49TB: 4min:30s (after raid initialization finished)
   - mkfs.ext4 on 15TB partition: 11min:52s (after raid initialization finished)
   - mkfs.ext4 on single 8TB partition can take more than 1h during raid initialization
   - can take hours or even whole day just for formatting whole disk space on a server
4. *mkdir* for mount point
5. edit */etc/fstab*
6. *mount*

repeat steps 4-6 if new OS gets installed.

Make sure the mount point is always the same or files will not be usable by DPM.

The above times were measured with 2TB disks used, imagine upgrading to 8TB or larger disks...

‘

## ZFS storage server setup

## ZFS storage server setup

1. creating and initialization of the raid array: *zpool create gridpool raidz3 sdb sdc sdd.........sdai sdaj spare sdak*

## ZFS storage server setup

1. creating and initialization of the raid array: *zpool create gridpool raidz3 sdb sdc sdd..........sdai sdaj spare sdak*

   - initializes raid array with 3 redundancy disks (raidz3) (similar to raid7) using 35 disks + 1 spare
   - makes a file system gridpool available
   - mounts file system gridpool under "*/gridpool*" (different mount point could be specified in the command)

# ZFS storage server setup

1. creating and initialization of the raid array: *zpool create gridpool raidz3 sdb sdc sdd..........sdai sdaj spare sdak*

   - initializes raid array with 3 redundancy disks (raidz3) (similar to raid7) using 35 disks + 1 spare
   - makes a file system gridpool available
   - mounts file system gridpool under "*/gridpool*" (different mount point could be specified in the command)

time needed: 10s (takes longer to put the command line together than to have the storage available and mounted)

To be fair, if you have a raid controller that doesn't support direct disk access, you need to create single disk raid0.
in our case: create 36 raid0 per machine using omconfig; takes about 2 min to finish

# ZFS storage server setup

1. creating and initialization of the raid array: *zpool create gridpool raidz3 sdb sdc sdd..........sdai sdaj spare sdak*

   - initializes raid array with 3 redundancy disks (raidz3) (similar to raid7) using 35 disks + 1 spare
   - makes a file system gridpool available
   - mounts file system gridpool under "*/gridpool*" (different mount point could be specified in the command)

<u>time needed:</u> 10s (takes longer to put the command line together than to have the storage available and mounted)

To be fair, if you have a raid controller that doesn't support direct disk access, you need to create single disk raid0.
in our case: create 36 raid0 per machine using omconfig; takes about 2 min to finish

after new OS is installed: *zpool import gridpool*

Mount point is stored in the file system and automatically the same again.

(No need to edit system files.)

# ZFS storage server setup

1. creating and initialization of the raid array: *zpool create gridpool raidz3 sdb sdc sdd..........sdai sdaj spare sdak*

- initializes raid array with 3 redundancy disks (raidz3) (similar to raid7) using 35 disks + 1 spare
- makes a file system gridpool available
- mounts file system gridpool under "*/gridpool*" (different mount point could be specified in the command)

time needed: 10s (takes longer to put the command line together than to have the storage available and mounted)

To be fair, if you have a raid controller that doesn't support direct disk access, you need to create single disk raid0.
in our case: create 36 raid0 per machine using omconfig; takes about 2 min to finish

after new OS is installed: *zpool import gridpool*

Mount point is stored in the file system and automatically the same again.

(No need to edit system files.)

It doesn't matter if you use 2TB or 8TB or even larger disks in the future!

## Performance comparisons

## Grid Storage workload

- copy in/out of files from/to worker nodes = sequential write/read access
- parallel access by different jobs
- input data usually large files
- output data can be very small (log files) or large too (data files)

# Performance tests

- read tests
  - 5.1TB of mixed files from ATLAS, LHCb, LSST, and CMS used
  - consist of input data and user output in 18,335 files
  - read content of files one after another (to /dev/null)
  - simulate parallel access by reading 10 files in parallel
  - ZFS uses default compression
- write tests of large files
  - write 100 files, 54GB each
  - for each file write 54GB, sync, rm
  - simulate parallel access by writing 5 files in parallel
  - ZFS without compression
- write tests of small files
  - write 200,000 files, 131kB each
  - like before but simulate parallel access by writing 20 files in parallel
- large parallel access test
  - 400 reads in parallel
  - all 100 writes for the 54GB files in parallel
  - 400 writes in parallel for the small files

# Test setup

- one of the machines with 3 external disk enclosures and 36 2TB disks used
- SL6 as OS since most Grid middleware still runs on SL6
- for the hardware raid controller the default configuration was used
  - write-back cache
  - adaptive read-ahead cache
- all filesystems where formatted with default options
  - changes, for example for XFS stripe size or number of disks for stripe, showed no large influence
  - in addition, for ZFS default compression (lz4) was enabled for the experiments data storage
- hardware raid: raid60 with 11 disks per raid6
- ZFS raid: 3 raidz2 in same pool (like raid60), 11 disks per raidz2
- measure performance for normal raid mode, degraded mode, and during rebuild
- for parallel access: access for group of files (10/5/20) is started, next group started after last file of previous group was read/written

System is not the most powerful and high-performance system, but that doesn't matter for comparisons.

Therefore focus on relative values and not on absolute values.

# Performance in normal raid mode

| | ZFS 3xraidz2 | XFS raid60 | Ext4 raid60 |
|---|---|---|---|
| read, 5.1TB, 18,335 files | | | |
| sequential read | 9,981s | 6,246s<br>-37% | 8,815s<br>-12% |
| 10 reads parallel | 7,164s | 9,936s<br>+39% | 10,036s<br>+40% |
| 100x(write of 54GB, sync, rm) | | | |
| seq. write | 7,849s | 8,296s<br>+6% | 15,902s<br>+103% |
| 5 writes in parallel | 8,133s | 17,596s<br>+116% | 39,460s<br>+385% |
| 200,000x(write of 131kb, sync, rm) | | | |
| seq. write | 2,453s | 10,232s<br>+317% | 10,816s<br>+341% |
| 20 writes in parallel | 348s | 1,651s<br>+374% | 1,274s<br>+266% |

Percentages are relative to ZFS values.

- sequential read time for XFS and Ext4 influenced by controller cache
- for parallel read access, controller cache doesn't help anymore
- ZFS shows much better performance especially for parallel access

# Performance in degraded raid mode

| | ZFS 3xraidz2 | XFS raid60 | Ext4 raid60 |
|---|---|---|---|
| read, 5.1TB, 18,335 files | | | |
| sequential read | 10,059s<br>+1% | 19,535s<br>+213% | 20,091s<br>+128% |
| 10 reads parallel | 7,568s<br>+6% | 19,094s<br>+92% | 17,949s<br>+79% |
| 100x(write of 54GB, sync, rm) | | | |
| seq. write | 7,828s<br>0% | 8,411s<br>+1% | 18,584s<br>+17% |
| 5 writes in parallel | 7,955s<br>-2% | 17,327s<br>-2% | 39,170s<br>-1% |
| 200,000x(write of 131kb, sync, rm) | | | |
| seq. write | 2,441s<br>+0% | 10,293s<br>+0% | 10,706s<br>-1% |
| 20 writes in parallel | 348s<br>+0% | 1,524s<br>-8% | 1,653s<br>+30% |

Percentages are relative to normal raid mode performance.

- read and write performance nearly unchanged for ZFS
- write performance for XFS unchanged, but decreased for Ext4
- XFS and Ext4 show much slower read performance in degraded mode than in normal mode

# Performance during rebuild

| | ZFS 3xraidz2 | XFS raid60 | Ext4 raid60 |
|---|---|---|---|
| read, 5.1TB, 18,335 files | | | |
| sequential read | 13,327s | 21,655s | 21,735s |
| | +34% | +247% | +147% |
| 10 reads parallel | 10,036s | 20,055s | 19,500s |
| | +40% | +102% | +94% |
| 100x(write of 54GB, sync, rm) | | | |
| seq. write | - | 15,126s | 23,127s |
| | - | +82% | +45% |
| 5 writes in parallel | - | 19,807 | 42,813s |
| | - | +13% | +8% |
| 200,000x(write of 131kb, sync, rm) | | | |
| seq. write | 4,210 | 9,584s | 11,974s |
| | +72% | -6% | +11% |
| 20 writes in parallel | 820s | 1,549s | 1,377s |
| | +136% | -6% | +8% |

Percentages are relative to normal raid mode performance.

- for reads, large decrease in performance for XFS and Ext4 and much smaller decrease for ZFS
- write performance for large files also decreased for XFS and Ext4, not measurable for ZFS due to short rebuild times
- rebuild time for hardware raid: 21h
- rebuild time for ZFS : 2h:15min
  (with 5.1TB of data in ZFS)
  (increases with amount of data stored, but restores redundancy while it works and not only when it finishes)
- for ZFS larger decrease in performance for small writes, but still much faster than XFS or Ext4

## Performance in normal raid mode

### for highly parallel read/write access

## Performance in normal raid mode

## for highly parallel read/write access

| | ZFS 3xraidz2 | XFS raid60 | Ext4 raid60 |
|---|---|---|---|
| read, 5.1TB, 18,335 files<br>400 reads in parallel | 5,253s<br>-27% | 21,512s<br>+117% | 23,400<br>+133% |
| 100x write of 54GB, sync, rm<br>100 writes in parallel | 10,800s<br>+33% | 27,751s<br>+58% | 46,468s<br>+18% |
| 200,000x write of 131kb, sync, rm<br>400 writes in parallel | 176s<br>-49% | 511s<br>-69% | 469s<br>-63% |

Percentages are relative to values for parallel access in normal raid mode.

- increased performance for parallel reads with ZFS, decreased performance for XFS and Ext4
- decreased performance for parallel writes, however ZFS still has better performance than XFS or Ext4 with only 5 files in parallel
- increased performance for parallel writes of small files for all file systems, ZFS about 3 times faster than XFS

# Different failure scenarios

- loose of single disk
  - rebuild using spare
- loose of server
  - connect disks to other server in any order
  - *zpool import POOL*
- loose of external enclosure
  - put disks in other enclosure to connect to server
  - *zpool import*
- silent data corruption on disk
  - ZFS uses check sums for every block
  - detects silent data corruption and repairs data when redundancy is available

With ZFS it doesn't matter in which order disks are connect to a server, it will find all disks belonging to a pool.
Different than a hardware raid system, ZFS provides real data protection.

# Edinburgh Grid storage setup

- 1 raidz3 with 32 data disks in it instead of the initial 2x raid6 setup
  - first DPM client server changed at end of 2015
  - after initial tests, moved all DPM client servers to ZFS
  - no ZFS related problems occurred

- with raidz3 about 30% slower reads than with 3xraidz2, while writes are at about the same rate

- real life example: draining a DPM client machine
  - drain: look up files in db on DPM head node, copy file out of the server, change path in db on head node, delete file on disk
  - running about 30 threads in parallel: network interface traffic on DPM client machine $>$ 9Gbps
  - DPM client machines have 10Gbps Ethernet interface

# Edinburgh Grid storage setup

- 1 raidz3 with 32 data disks in it instead of the initial 2x raid6 setup
  - first DPM client server changed at end of 2015
  - after initial tests, moved all DPM client servers to ZFS
  - no ZFS related problems occurred

- with raidz3 about 30% slower reads than with 3xraidz2, while writes are at about the same rate

- real life example: draining a DPM client machine
  - drain: look up files in db on DPM head node, copy file out of the server, change path in db on head node, delete file on disk
  - running about 30 threads in parallel: network interface traffic on DPM client machine $>$ 9Gbps
  - DPM client machines have 10Gbps Ethernet interface

Raidz3 fast enough for our use case while providing better redundancy and more storage capacity than the initial raid6 setup.

1 raidz3 per machine = only 4 disks used for redundancy and spare providing 32 data disks per machine.

64TB extra capacity (more than one of our servers with 2 TB disks in it)

More extra capacity through compression possible...

## ZFS Compression

- built-in compression at the block level
- different compression algorithms available
- default compression algorithm (lz4) shows no performance impact and can always be enabled
- different algorithms show no large difference in compression for data files of the LHC experiments
- we get an overall compression of 5% with lz4 (mainly determined by ATLAS files, could change in the future with supporting other VOs )

5% equals about 55TB extra capacity = extra capacity of one of our servers with 2TB disks for free and without additional server hardware

## Conclusion

- reliable working of ZFS on Linux on our DPM storage servers without any problems since about 1 year

- much easier and simpler administration without the need to change OS system files

- ZFS showed much better performance than XFS and Ext4, especially for parallel access patterns

- future-proof by using raid configurations with 3 redundancy disks

- built-in block level compression can reduce used storage space
  works for LHC experiments data storage, and can be even more important when supporting other VOs

- reduced costs possible by not using hardware raid controllers and combining disks on different controllers
  also due to other effects like compression

- ZFS on Linux in active development

# More information about ZFS on Linux and ZFS for DPM storage

- ZFS on Linux main web page: `http://zfsonlinux.org/`
- blog of the GridPP storage group: `http://gridpp-storage.blogspot.co.uk/`
  - comparisons of hardware raid and ZFS
  - ZFS compression
  - setup of a ZFS on Linux storage server for GridPP
  - direct links to ZFS related GridPP storage posts: `http://ebert.homelinux.org/blogposts.html`
- ZFS talk by Bill Moore and Jeff Bonwick, main ZFS architects

  `https://www.cs.utexas.edu/users/dahlin/Classes/GradOS/papers/zfs_lc_preso.pdf`

  `http://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf`
- other presentations about ZFS and ZFS on Linux

  `http://www.slideshare.net/mewandalmeida/zfs`

  `http://www.slideshare.net/Clogeny/zfs-the-last-word-in-filesystems`

  `http://indico.cern.ch/event/518392/contributions/2195790`

  `https://indico.cern.ch/event/539135/contributions/2214537/`

  `http://indico.cern.ch/event/518392/contributions/219579`

## Thank you!

## Backup Slides

# raidz3 vs 3xraidz2

| | 3xraidz2 normal mode | raidz3 normal mode | 3xraidz2 degraded mode | raidz3 degraded mode | 3xraidz2 during rebuild | raidz3 during rebuild |
|---|---|---|---|---|---|---|
| read, 5.1TB, 18,335 files | | | | | | |
| sequential read | 9,981s | 12,675s +27% | 10,059s | 12,651s +26% | 13,327s | 16,930s +27% |
| 10 reads parallel | 7,164s | 9,846s +37% | 7,568s | 10,101s +33% | 10,036s | 12,837s +28% |
| 100x(write of 54GB, sync, rm) | | | | | | |
| seq. write | 7,849s | 8,171s +4% | 7,828s | 8,588s +10% | – | 25,778s |
| 5 writes in parallel | 8,133s | 8,382s +3% | 7,955s | 8,470s +6% | – | 25,581s |
| 200,000x(write of 131kb, sync, rm) | | | | | | |
| seq. write | 2,453s | 2,459s +0% | 2,441s | 2,461s +1% | 4,210s | 3,556s -16% |
| 20 writes in parallel | 348s | 373s +7% | 348s | 374s +7% | 820s | 619s -25% |

## ZFS compression

- random selection of ATLAS data files on site (we have mostly Atlas data files on site)

- all data files on site used for the other VOs

| compression algorithm | ATLAS 4.3TB | LHCb 0.5TB | CMS 0.2TB | ILC 0.5TB | LSST 1TB | HyperK 0.7TB |
|---|---|---|---|---|---|---|
| lz4 | 1.04 | 1.03 | 1.01 | 1.00 | 1.18 | 1.02 |
| lzjb | 1.02 | 1.01 | 1.00 | 1.00 | 1.16 | 1.01 |
| gzip | 1.05 | 1.03 | 1.01 | 1.00 | 1.24 | 1.03 |
| gzip1 | 1.05 | 1.03 | 1.01 | 1.00 | 1.21 | 1.03 |
| gzip5 | 1.05 | 1.03 | 1.01 | 1.00 | 1.23 | 1.03 |
| gzip9 | 1.05 | 1.03 | 1.01 | 1.00 | 1.24 | 1.03 |

- default compression shows good results by much lower overhead than all other algorithms

- no performance difference between default compression and no compression

- it doesn't hurt to keep compression on

- would be good if sites providing storeage for VOs other than ATLAS could do compression tests too

## ZFS compression

- random selection of ATLAS data files on site (we have mostly Atlas data files on site)

- all data files on site used for the other VOs

| compression algorithm | ATLAS 4.3TB | LHCb 0.5TB | CMS 0.2TB | ILC 0.5TB | LSST 1TB | HyperK 0.7TB |
|---|---|---|---|---|---|---|
| lz4 | 1.04 | 1.03 | 1.01 | 1.00 | 1.18 | 1.02 |
| lzjb | 1.02 | 1.01 | 1.00 | 1.00 | 1.16 | 1.01 |
| gzip | 1.05 | 1.03 | 1.01 | 1.00 | 1.24 | 1.03 |
| gzip1 | 1.05 | 1.03 | 1.01 | 1.00 | 1.21 | 1.03 |
| gzip5 | 1.05 | 1.03 | 1.01 | 1.00 | 1.23 | 1.03 |
| gzip9 | 1.05 | 1.03 | 1.01 | 1.00 | 1.24 | 1.03 |

(files in the 4.3TB sample)

| Atlas data file type | compression with lz4 |
|---|---|
| tgz files | 1.24 |
| RDO files | 1.07 |
| ESD files | 1.00 |
| EVNT files | 1.00 |
| HITS files | 1.03 |
| AOD files | 1.02 |

- tgz is surprising
- but even root files can still be compressed
- but low statistics
- let's have a look to larger sample...

# Compression for ATLAS files

| | number of files | *du –apparent-size* report | *du* report | ratio |
|---|---|---|---|---|
| tgz | 1.1M | 2.7TB | 2.5TB | 1.08 |
| RDO | 1.1k | 1.1TB | 0.87TB | 1.26 |
| ESD | 44k | 5.6TB | 4.9TB | 1.14 |
| EVNT | 52k | 11.7TB | 11.6TB | 1.01 |
| HITS | 120k | 34.6TB | 34.6TB | 1.00 |
| AOD | 346k | 401.5TB | 388.7TB | 1.03 |
| other | - | 215.1TB | 205.8TB | 1.05 |

- tgz compression varies between 1.00 and 2.00
- ”other” is everything stored for ATLAS that doesn’t fit in the above categories, for example user analysis output files
- we get an overall compression of 5% (mainly determined by ATLAS files, could change in the future with supporting other VOs)
  - 5% equals about 55TB extra capacity = extra capacity of one of our old servers for free and without additional server hardware

# Data protection in RAID6+HS

- Writes go to all disks
- Reads come from data disks only
- When a disk is unavailable, read remaining data disks+ redundancy disk and rebuild broken disk from all remaining disks using HS
  - Needs to read all disk blocks on all disks, even if empty
    100TB raid, 10% full, with 8TB disks: read 100TB, write 8TB
  - Rebuild always takes very long time
  - Data only available once the rebuild finished
- Protect against disk failures and makes sure the application gets data
- Can not protect against silent data corruption!
  - It doesn't know if the data read is the data the application sent to disk
  - If wrong data is read, no way to correct it even if it could use the redundancy in the RAID

There is not really data protection with hardware raid systems...

## Data protection in raidz2+HS

- Writes go to as many disks as needed
  - variable block size/stripe length depending on file size
- Reads come from data disks only
- When a disk is unavailable, read remaining data disks + redundancy disk and rebuild broken disk from all remaining disks using HS
  - Only needs to read blocks with live data

    100TB raid, 10% full, with 8TB disks: read only 10TB, write 0.8TB
  - Rebuild goes through the file system starting from root directory
  - With ongoing rebuild time, more and more redundancy is restored
- Protect against disk failures and makes sure the application gets data
- Can also protect against data corruption
  - Checksum calculated for each block and stored in parent block
  - Each read is compared against it's checksum to make sure the correct data is read
  - ZFS can detect wrong data is read, reconstruct correct data from redundancy and correct wrong data blocks
- ZFS never overwrites live data (COW), file system always consistent, never needs fsck
  - Random writes $\longrightarrow$ sequential writes

    Also reorders different write requests

# Tests at CERN

- Write $\sim$ 2GB file with specific pattern in it, read it back, compare patterns
  - Deployed on 3000 nodes, run every 2h
  - After 5 weeks, 500 errors on 100 nodes
    - Single bit errors: 10%
    - Sector or page sized region errors: 10%
    - Bug in WD disk firmware: 80%
- Hardware raid verify run on 492 systems (1.5PB) over 4 weeks
  - ”fix” of 300 block errors
  - However, hardware raid doesn’t know if data or parity block is wrong
- Read normal files back from disk and compare to checksum stored on tape
  - 33,700 files checked (8.7TB)
  - 22 mismatches found
  - 1 out of 1500 files bad
  - (at least) 1 error every 400GB
  - Byte error rate of $3 \times 10^{-7}$

https://indico.cern.ch/event/13797/contributions/1362288/attachments/115080/163419/Data_integrity_v3.pdf

back in 2007 by Dr. Bernd Panzer-Steindel

Was done before and later with similar results, also regularly done by companies like IBM

## Why not a traditional storage system

- No data awareness
  - Rebuilds need to read always all disks and write full new disk, no matter how much data is stored
  - Rebuild can take many days, and it's getting worst with larger disks in the future
    With long rebuild times, it's more likely to have more failed disks before redundancy is fully restored $\Rightarrow$ RAID5 dead for a reason, soon RAID6 will be too
- No data protection, only disk space protection
  - If a block can be read then it is delivered, no matter if what is read is correct or not
  - No way to correct silent data corruption, can't even detect it
- Long and complicated process to setup storage before usable in the OS
  - Will get worst in the future with larger disks
- File system on top can easily get confused in failure situations
- Strong connection between OS system config and storage availability
  - In case of upgrades/OS disk failures, system config needs to be redone to use storage
- Not easy to upgrade storage based on hardware raid systems