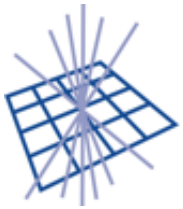
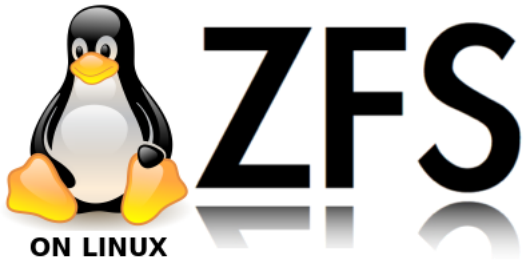
THE UNIVERSITY of EDINBURGH



GridPP
UK Computing for Particle Physics

HEPiX Fall 2016 Workshop



based on Edinburgh/ScotGrid Experience

Marcus Ebert
University of Edinburgh
marcus.ebert@ed.ac.uk

- ZFS/ZFS on Linux
- Performance and compression tests
- ScotGrid installations
- Experience and feature set

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
- available in repositories for many Linux distributions
  - **repositories for RH/SL/CentOS exists, DKMS and kABI-tracking kmod style**
  - **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 with snapshot features
- COW file system that transforms random writes into sequential writes
- protection against silent data corruption and self healing of data
- supports 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
- local file system like xfs, ext4, ...

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:
  - 3xR610: 12GB RAM, 8/16C E5620 2.4GHz
    PERC H800, 36x2TB NL6 7.2k, in DAS (3 external enclosures)
    RAID6, 17 disks + 1 HS (2x)
  - 3x R510, 24GB RAM, 8/16C E5620 2.4GHz
    36x2TB NL6 7.2k, 12xinternal (H700) and 24 in 2 external enclosures (H800)
    RAID6, 11 disks + 1 HS and 23 disks + 1HS
    2 machines were updated with 17 8TB disks each
  - 5xR720xd, 64GB RAM, 12/24C E5-2620v2 2.1GHz
    36x4TB NL6 7.2k, 12xinternal (H710)and 24 in 2 external enclosures (H810)
    RAID6, 11 disks + 1 HS and 23 disks + 1HS
- disks on different controllers can't be combined
  previously each machine with 2x(raid6+1HS) resulting in 30 data disks per machine

Performance comparisons

ZFS 3xraidz2 vs Hardware Raid 60 + Ext4/XFS

for Grid Storage use case

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

Performance comparison results

- normal raid mode
  - for read of single files, controller cache reduces read times for xfs/ext4
  - for parallel read/write access, ZFS shows much better performance
  - ZFS also shows better performance for writes
- degraded raid mode
  - ZFS shows (nearly) unchanged read/write performance
  - XFS/Ext4 showed largely decreased read performance
- during rebuild
  - large decrease in read performance for XFS/Ext4, only small decrease for ZFS performance
  - write performance for small files decreased for ZFS, but still faster than XFS/Ext4 by a factor of 1.7...2.8
  - much faster rebuild with ZFS than with hardware raid

    (depends on amount of data stored in the system for ZFS)
- for highly parallel access, ZFS showed best performance, sometimes better than XFS with less parallel access

details about the test results and compression rates:

`https://indico.cern.ch/event/505613/contributions/2230928/attachments/1347288/2041281/oral-final-533.pdf`

ZFS Compression

- built-in compression at the block level
- different compression algorithms available
- default compression algorithm (lz4 in Linux port) 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)

ZFS Compression

compression algorithm	ATLAS 4.3TB	LHCb 0.5TB	CMS 0.2TB	ILC 0.5TB	LSST/DES 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

- 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
- 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
- need tests from sites providing storage for VOs other than ATLAS

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

Edinburgh usage of ZFS on Linux systems for GridPP

New storage file system

- use 2x (ZFS raidz2 + HS) for machines with the new 8TB disks
  - one for the 8TB disks, one for the 2TB disks
- Use ZFS raidz3 + HS for machines with single kind of disks
  - 4x 2x2TB + 5 x2x4TB = 56 TB extra space compared to 2x(raid6+HS)
  - better redundancy than with raid6/raidz2

Due to use of 1000 vs 1024: 8TB→7.3TB; 4TB→3.7TB; 2TB→1.8TB

- Total disk space managed by ZFS: 1,035TB
  (useable space, without compression taken into account)
  - On 11 machines and 396 disks (182x2TB disks + 180x4TB + 34x8TB disks)

ZFS as storage file system

- one file system per zpool
  - use weights in DPM to account for different servers instead of number of file systems

- enabled ZFS compression
  - default compression algorithm on Linux is lz4
  - no visible bad impact on performance
  - reduced space usage depending on VO

- use *xattr=sa*

- very easy storage administration
  - no more *omconfig* + *fdisk* + *mkfs* + *nano /etc/fstab* + *mkdir $MOUNTPOINT* + *mount*
  - just a single command: *zpool create tank raidz3 ....*
  - instead of many hours/days, it just takes seconds until the whole storage is available and mounted
  - no more changes in OS system files needed

Edinburgh Grid storage experience

- first DPM client server changed at the end of 2015
- after initial tests, moved all DPM client servers to ZFS
- started with v0.6.5.3, now on v0.6.5.7/v0.6.5.8
- no ZFS related problems occurred
- 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 transfer rate 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.

Other ZFS GridPP usage in Edinburgh

- HA server system for GridPP middleware
- middleware runs in VMs on both servers
- storage part managed by ZFS and DRBD on SL7
  - 2 data disks per machine, create ZFS mirror on each server
  - skip POSIX layer, and create 2 block devices (zvol)
  - use these block devices in DRBD, one primary on each server
  - VM images are on the drbd managed space

Other use cases implemented by ScotGrid

Edinburgh GridPP group manages Grid middleware and storage only;
Glasgow and Durham have more flexibility

GridPP storage in Glasgow

- test mode for DPM storage
  - supporting different VOs
  - having one DPM pool each of Atlas, LHCb, others
  - creating one ZFS for each DPM pool per server
  - ZFS size freely changeable within pool limits using refquotas/reservations
    $\Rightarrow$ freely adjust size of DPM pools depending on demand

ZFS at Durham

- planning to move everything to ZFS
- any new server will be HBA only and run on ZFS
- some services already migrated
  - $HOME for local users/desktops
    - 10x(2x2TB) ZFS mirror pool, compressratio: 1.97
    - all-flash server will be purchased this week to replace home server
    - configuration for all-flash server needs to be tested
  - Grid space for user output/software
    - 12 disk raidz2, compressratio: 2.97
    - mostly for Pheno users to store job submission tools and job output
  - ZFS based backup system
    - snapshot based
    - *zfs send/receive* using compression and incremental send features
    - send zfs' from live server to backup server, not tape
    - backup server has latest live ZFS mirrored + backups (snapshots) from previous days/weeks
  - OpenStack backend to combine disks in redundant way

ZFS on Linux, features & observations

- stable operations in all versions used
  - provides better redundancy than hardware raid we had before (raidz3, 3-way mirror)
  - snapshots
  - supports large block/record size
    (512bytes to 16M, 128k default, enabled large blocks limited to 1M by default)
  - NFS sharing feature works fine
  - zfs send/receive with compression and incremental send
- ZFS Event Daemon (zed)
  - handles communications and actions in different failure cases
  - adds *zpool events* similar to *zpool history*
  - sends emails out for failures/errors or for all action on the pool (verbose)
  - simple config file under /etc, actions handles by shell scripts
  - works fine except using hot spare automatically in case of disk failure
    * to be tested on new LSST servers with HBA only
    * sending out emails automatically works
    * changing script for automatic handling of HS would easily be possible

ZFS on Linux, features & observations

- zdb available too

- most features known from Solaris work
  - *zpool iostat*
  - *zpool history*
  - automatic nfs share of snapshots
  - ...

- some things are not implemented yet
  - iSCSI share
  - ClamAV for Virus scan
  - delegated permissions (base system already in v0.7rc1)

- many parameters can be tuned under */sys/module/zfs/parameters/*

Conclusion

- reliable working of ZFS on Linux on our DPM storage servers without any problem over the last 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 or 3-way mirror
- 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, no need for Solaris based OS

  also due to other effects like compression
- ZFS on Linux in active development

Traditional Disk Storage Administration

http://www.iro.umontreal.ca/~dift3830/ZFS.pdf

But with ZFS....

http://www.iro.umontreal.ca/~dift3830/ZFS.pdf

Also true with ZFS ON LINUX

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 all 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/`

  `https://indico.cern.ch/event/505613/contributions/2230928/`

Thank you!

Backup Slides

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 3×10^{-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 ⇒ 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