Operational characteristics of a ZFS-backed Lustre filesystem

Daniel Kobras

science + computing ag

IT-Dienstleistungen und Software für anspruchsvolle Rechnernetze

Tübingen | München | Berlin | Düsseldorf
Use Case
Environment

- Lustre 1.8 filesystem on JBOD hardware runs out of life
  - ldiskfs on top of mdraid
  - whacky hardware (frequent drive failures)
  - abandoned storage management software (no-go with compliant Java versions)
  - performance issues unless using Sun/Oracle Lustre kernel (stuck with 1.8.5)
Environment

• Re-use storage and server hardware for scratch filesystem
  • upgrade to current OS and Lustre versions
  • avoid mdraid layer
  • avoid obsolete JBOD management
  • increase drive redundancy

-> Lustre (2.5.3) with ZFS backend using raidz3 (8+3) vdevs

-> Learn from previous installation and apply overall improvements, not specifically tied to ZFS
Administration & Maintenance
Deployment

• Looser dependency on kernel version (facilitates security updates)
• Using spl and zfs auto-build of modules via dkms
• Backported support for zfs/spl 0.6.4 (LU-6038) to Lustre 2.5.3
• Step-by-step creation:
  • Create ZPool
  • Test with temporary ZFS posix layer
  • Zap test filesystem
  • Set `canmount=off`
  • Create Lustre backend filesystems
Stability

• Server OOM with zfs/spl 0.6.3 during iozone rewrite test
• Clients crashed due to empty ACLs (otherwise filtered out by ldiskfs, LU-5150)
• ZFS uncovered several faulty drives (previously undetected/unidentified by mdraid)
JBOD storage

• There's life beyond crappy Java management tools
• There's standard tools and even a SCSI standard for talking to storage enclosures
  • `sg_ses` and `/sys/class/enclosure` FTW!
  • Easy, automated matching of `/dev/sdX` to corresponding drive slot
  • Steering locator LEDs from OS, no need for external tools
• `/dev/disk/by-vdev` makes it even easier:
  • ZFS reports failed drive in enclosure X, slot Y
    -> location immediately visible from zpool status output
    (optimize disk names for most common daily tasks, ie. open calls for failed drives, replace failed drives)
• Less impact from resilvering compared to md raid rebuild
High Availability

• Pacemaker integration
  • Split JBOD drives into 'left' and 'right' pools, controlled by either frontend server
  • Separate resources for each ZPool and each Lustre target
• Customized scripts for more finegrained control than supplied init script
  • ZFS: ZPool import with `cachefile=none` (no MMP equivalent, requires fencing)
  • Filesystem: modified to grok ZFS pool syntax
Debugging
zdb

• Under-the-hood debugging of ZFS/ZPool internals
• Almost, but not quite, entirely unlike debugfs
• User interface apparently designed by Huffman-coding a standard set of options onto a Morse alphabet
• Calling `zdb  -ddddd` is perfectly normal, and not to be mistaken with `zdb  -dddd`
zdb – Example

• Find physical location of data in file 'x':

  # ls -i x
  12345678 x

  # zdb -dddddd tank/fs1 12345678

Dataset tank/fs1 [ZPL], ID 45, cr_txg 19, 251M, 3292 objects, rootbp DVA[0]=<3:59b6000:800>
DVA[1]=<0:5d67800:800> [L0 DMU objset] fletcher4 lz4 LE contiguous unique double size=800L/200P
birth=1626187L/1626187P fill=3292
cksum=154bf6b90b:7136427a15d:147c0807d0853:2a4d1fac6823a
(tbc)
zdb – Example

# zdb -ddddd tank/fs1 12345678

(cont.)

Object  lvl .iblk  dblk  dsize  lsize  %full  type
12345678  2  16K  128K  259K  256K  100.00  ZFS
plain file
(...)

Indirect blocks:

0  L1  3:59b0000:800 4000L/200P  F=2  B=1626187/1626187
0      L0  3:5956800:2c000 20000L/20000P  F=1

B=1626187/1626187

20000  L0  3:5982800:2c000 20000L/20000P  F=1

B=1626187/1626187
zdb – Examples

• Data virtual address (DVA):  <vdev>:<offset>:<size>
• Data at L0 DVAs (if F=1)
• Map DVA to physical location on disk:
  • Easy for simple vdev types (single drive, mirror):
    block = (offset + headersize) / blocksize, with headersize = 4*1024*1024 (4MB)
  • For raidz{,2,3}: Read the source
  • Or just zdb -R tank <DVA> (if you still can)
  • Or cheat with strace zdb -e -R tank <DVA> (on exported test pool; analyze disk access)
Performance Characteristics
Performance Baseline

- Study performance of Lustre client
- Relate to performance of local ZFS (ZPL)
- Study FS behaviour with more general I/O patterns, not just bulk I/O
- Study performance in relation to capabilities of storage hardware:
  - Interconnect performance
  - HDD performance
HDD Performance

 throughput
 block size
 char. size
HDD Performance

![Graph showing HDD performance with block size on the x-axis and throughput on the y-axis. The graph illustrates the relationship between block size and throughput for random and bulk operations. The graph also indicates a char. size region where performance is affected.]
HDD Performance

• HDD datasheet:
  • 1 TB capacity
  • 7200 RPM
  • 8-9 ms seek latency
  • 90 MB/s sequential throughput

• Characteristic values:
  • Max. rotational latency = 1 / RPM = 8.3 ms
  • Random IOPS =
    1s / (seek latency + 1/2 rotational latency) = 79
  • Seq. throughput / Random IOPS = 1.1 MB char. I/O size
Performance Expectations

• Configuration:
  • 6x vdev raidz3 (8+3)
  • 128k recordsize
  • No compression

• Bulk throughput:
  • Writes need to transfer parity from host to disk
  • Max. write throughput = 8/11 x max. read throughput

• Per vdev:
  • Expected char. I/O size for 8+3 raidz3:
    8 x 1.1 MB = 8.8 MB
  • Might saturate at smaller I/O size due to interconnect limit
Benchmark Results (ZPL)

ZFS Posix Layer (IOZone results)

16 GB test file size

throughput (kB/s)

rec (kB)
Benchmark Results (Lustre Client)

Lustre-Client (IOZone results)

128 GB test file size

different scale!
Results

• ZPL:
  • RMW overhead for random I/O < recordsize
  • Significant metadata overhead for write operations (write vs. rewrite, rewrite vs. read)
  • For I/O units >= recordsize, random write performance similar to bulk
  • Read performance also depends on write pattern, IOPS bound for I/O units < 4-8 MB
  • In some tests, weird things happen at 4 MB I/O size

• Lustre:
  • Glosses over ZPL small I/O performance drawbacks
  • 2.5.3: consistent but slow single-stream performance
  • (not shown here) Bulk I/O saturates interconnect
Conclusion

• ZFS (and OS update) provide stable configuration on previously unstable hardware
• Easier administration and maintenance than mdraid-based setup
• Well suited for JBOD storage
• Different backend FS uncovered bug in Lustre, but otherwise stable
• ZFS performance meets expectations
• Lustre performance
  • ok for most I/O loads with units >= 128kB
  • single-stream performance generally slow, considering upgrade to later Lustre versions
Vielen Dank für Ihre Aufmerksamkeit.

Daniel Kobras
science + computing ag
www.science-computing.de

Telefon 07071 9457-0
info@science-computing.de