Linux Filesystem & Storage Tuning

Christoph Hellwig

LST e.V.

LinuxCon North America 2011

Introduction

The examples in this tutorial use the following tools:

- e2fsprogs
- xfsprogs
- mdadm

Overview

Checklist for filesystem setups:

1. Analyze the planned workload
2. Choose a filesystem
3. Design the volume layout
4. Test
5. Deploy
6. Troubleshoot

Filesystem workloads

A few rough workload characteristics are very important for the filesystem choice and volume setup:

- Data vs Metadata proportion
- Sequential or random I/O
- I/O sizes
- Read vs write heavy

Filesystem choice

ext4 Improved version of the previous ext3 filesystem. Most advanced derivative of the Berkeley FFS, ext2, ext3 family heritage.

- Good single-threaded metadata performance
- Plugs into the ext2, ext3 ecosystem

XFS Big Data filesystem that originated under SGI IRIX in the early 1990ies and has been ported to Linux.

- Lots of concurrency by design
- Design for large filesystems, and high bandwidth applications

Data layout

Basic overview of disk layout choices

	throughput	IOPS
no redundancy	striping	concatenation
single redundancy	RAID 5	concatenation + mirroring
double redundancy	RAID 6	concatenation + triple mirroring

Data layout - external log device

The log or journal is used to keep an intent log to provide transaction guarantees.

- Write-only except for crash recovery
- Small, sequential I/O
- Synchronous for fsync-heavy applications (Databases, NFS server)

For many use cases moving the log to a separate device makes improves performance dramatically.

Data layout - external log device (cont.)

- The log device also needs mirroring
- Choice of device: disk, SSD
- Does generally not help if you already have battery backed cache

Mdadm - Intro

RAID 1:

```
$ mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sd[bc]
mdadm: Note: this array has metadata at the start and
    may not be suitable as a boot device.  If you plan to
    store '/boot' on this device please ensure that
    your boot-loader understands md/v1.x metadata, or use --metadata=0.90
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

RAID 5:

```
$ mdadm --create /dev/md1 --level=5 --raid-devices=4 /dev/sd[defg]
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md1 started.
```

Mdadm - Advanced Options

Useful RAID options

name	default	description
-c / –chunk	512KiB	chunk size
-b / –bitmap	none	use a write intent bitmap
-x / –spare-devices	0	use nr devices as hot spares

Note: at this point XFS really prefers a chunk size of 32KiB.

```
mdadm --create /dev/md1 --level=6 --chunk=32 \
      --raid-devices=7 --spare-devices=1 /dev/sd[defghijk]
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md1 started.
```

Tip of the day: wiping signatures

To wipe all filesystem / partition RAID headers:

```
$ dd if=/dev/zero bs=4096 count=1 of=/dev/sdl
$ wipefs -a /dev/sdl
```

Creating XFS filesystems

```
$ mkfs.xfs -f /dev/vdc1
meta-data=/dev/vdc1              isize=256    agcount=4, agsize=2442147 blks
         =                       sectsz=512   attr=2, projid32bit=0
data     =                       bsize=4096   blocks=9768586, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=4769, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

- The -f option forces overwriting existing filesystem structures

Mkfs.xfs advanced settings

Useful mkfs.xfs options

name	default	maximum	description
-l size	$\frac{1}{2048}$	2g	size of the log
-l logdev	internal	-	external log device
-i size	256	2048	inode size
-i maxpct	25 / 5 / 1	0	% of space used for inodes
-d agcount	4	$2^{32} - 1$	nr of allocation groups

```
$ mkfs.xfs -f /dev/vdc1 -l logdev=/dev/vdc2,size=512m -i size=1024,maxpct=75
meta-data=/dev/vdc1              isize=1024   agcount=4, agsize=2442147 blks
         =                       sectsz=512   attr=2, projid32bit=0
data     =                       bsize=4096   blocks=9768586, imaxpct=75
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =/dev/vdc2              bsize=4096   blocks=131072, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

Tip of the day: xfs_info

The xfs_info tool allows to re-read the filesystem configuration on a mounted filesystem at any time:

```
$ xfs_info /mnt
meta-data=/dev/vdc1              isize=256    agcount=4, agsize=2442147 blks
         =                       sectsz=512   attr=2, projid32bit=0
data     =                       bsize=4096   blocks=9768586, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=4769, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

Creating ext4 filesystems

```
$ mkfs.ext4 /dev/vdc1
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
2444624 inodes, 9768586 blocks
488429 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=0
299 block groups
32768 blocks per group, 32768 fragments per group
8176 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000, 7962624

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 35 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

Creating ext4 filesystems (cont.)

Make sure to always disable automatic filesystem checks after N days or reboots:

```
$ tune2fs -c 0 -i 0 /dev/vdc1
tune2fs 1.41.12 (17-May-2010)
Setting maximal mount count to -1
Setting interval between checks to 0 seconds
```

External logs need to be initialized before the main mkfs:

```
$ mkfs.ext4 -O journal_dev /dev/vdc2
```

Mkfs.ext4 advanced settings

Useful mkfs.ext4 options

name	default	maximum	description
-J device	internal	-	external log device
-J size	32768 blocks	102,400 blocks	size of the log
-i	1048576	-	bytes per inode
-I	256	4096	inode size

Filesystem stripe alignment

Filesystems can help to mitigate the overhead of the stripe r/m/w cycles:

- Align writes to stripe boundaries
- Pad writes to stripe size

XFS stripe alignment

Let's create an XFS filesystem on our RAID 6 from earlier on:

```
$ mkfs.xfs -f /dev/md1
meta-data=/dev/md1               isize=256    agcount=32, agsize=9538832 blks
         =                       sectsz=512   attr=2
data     =                       bsize=4096   blocks=305242624, imaxpct=5
         =                       sunit=8      swidth=40 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=149048, version=2
         =                       sectsz=512   sunit=8 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

Important: sunit=8, swidth=40 blks

- The RAID chunk size is 32KiB, the filesystem block size is 4KiB
  - 32/4 = 8 (Stripe Unit)
- We have 8 devices in our RAID 6. 1 Spare, 2 Parity
  - 8 - 1 - 2 = 5 (Number of Stripes)
  - 5 * 8 = 40 (Stripe Width)

XFS stripe alignment (cont.)

For hardware RAID you'll have to do that math yourself.

```
$ mkfs.xfs -f /dev/sdx -d su=32k,sw=40
meta-data=/dev/sdx               isize=256    agcount=4, agsize=15262208 blks
         =                       sectsz=512   attr=2
data     =                       bsize=4096   blocks=61048828, imaxpct=25
         =                       sunit=8      swidth=320 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=29808, version=2
         =                       sectsz=512   sunit=8 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

Note: -d su needs to be specified in byte/kibibyte, not in filesystem blocks!

Ext4 stripe alignment

With recent mkfs.ext4 ext4 will also pick up the stripe alignment, or you can set it manually:

```
$ mkfs.ext4 -E stride=8,stripe-width=40 /dev/sdx
```

But at least for now these values do not actually change allocation or writeout patterns in a meaningful way.

Mount options

In general defaults should be fine, but there are a few exceptions.

Mounting XFS filesystens with external logs:

```
$ mount -o logdev=/dev/vdc2 /dev/vdc1 /mnt/
[33369.618462] XFS (vdc1): Mounting Filesystem
[33369.658128] XFS (vdc1): Ending clean mount
```

Inode64

By default XFS only places inodes into the first TB of the filesystem, ensuring that inode numbers fit into 32 bits.

- only require if you have apps using old system calls
- causes performance problems due to bad locality on large filesystem
- can cause unexpected **ENOSPC** errors when creating files

Unless you are using proprietary backup software from the 1990s you are probably safe using the inode64 mount option:

```
$ mount -o inode64 /dev/vdc1 /mnt/
[33369.618462] XFS (vdc1): Mounting Filesystem
[33369.658128] XFS (vdc1): Ending clean mount
```

The barrier saga

- XFS and ext4 both default to flushing the disk write cache if one is present
- Changing this is risky, and will lead to data loss if done incorrectly

The option to turn off flushing a volatile write cache is called **nobarrier** for historical reasons. For even more historical reasons it can also be **barrier=0** on ext4.

The barrier saga (cont.)

When is it safe to turn off the cache flushes (aka **nobarrier**)?

- Only if you know that all volatile write caches are turned off

Sometimes you have to disable volatile caches to get a safe operation:

- If using software RAID / LVM before ca Linux 2.6.37
- If using external logs before circa Linux 3.0

Disabling write caches also is beneficial for various workloads if not required.

Volatile write caches on SATA disk

Query:

```
$ hdparm -W /dev/sda

/dev/sda:
 write-caching =  1 (on)
```

Modify:

```
$ hdparm -W 0 /dev/sda

/dev/sda:
 setting drive write-caching to 0 (off)
 write-caching =  0 (off)
```

Note: hdparm cache settings are not persistent over a reboot

Volatile write caches on SAS / FC disk

Query:

```
sdparm --get WCE /dev/sdx
  /dev/sdx: ATA       SEAGATE ST32502N   SU0D
WCE           1   [cha: y]
```

Modify:

```
$ sdparm --clear WCE /dev/sdx
  /dev/sdx: ATA       SEAGATE ST32502N   SU0D
```

Note: sdparm settings can be made persistent using –save **if** the disk supports it.

I/O schedulers

Linux has three I/O schedulers: cfq, deadline, noop.

- In most distributions cfq is the default
- It is a very bad default except for single-SATA spindle desktops

The quick fix:

```
$ echo deadline > /sys/block/sda/queue/scheduler
```

In fact noop might be even better for many workloads, but deadline is a safe default.

Making CFQ not suck

```
echo 0 > /sys/block/sda/queue/iosched/slice_idle
```

This remove the idling on different classes of request. Which means that you'll actually be able to get closer to maxing out the hardware.

Fragmentation

In general ext4 and XFS do not have fragmentation problems.

In fact running the defragmentation tool can cause "fragmentation" problems.

- The defragmentation tools optimize the extent map in a file
- On full filesystems that fragments the free space

Fragmentation (cont.)

So when should I defragment?

- When a single file contains far too many extents
- Typically caused by workloads that randomly write into large sparse files:
  - Hadoop
  - Bittorrent clients
  - VM images
  - Some HPC workloads

Most of these could be trivially fixed by preallocating the file, or using the extent size hint on XFS.

Finding and fixing fragmentation - XFS

The xfs_bmap tool can be used to check the number of extents of a file:

```
$ xfs_bmap /home/qemu-data.img | grep -v hole | wc -l
2014
```

The threshold for considering a file fragmented would be more than 1 extent per about 100MB of file data. Badly fragmented is one extent for less than 10MB of data.

To fix the fragmentation run the xfs_fsr tool:

```
$ xfs_fsr /home/qemu-data.img
$ xfs_bmap /home/qemu-data.img | grep -v hole | wc -l
9
```

Alternatively run it with a device file as argument to pass over a whole filesystem.

Further information

- http://xfs.org/index.php/XFS_Papers_and_Documentation

Has an XFS Users Guide and XFS Training Labs for a multi-day introduction to understanding and setting up XFS with details hand-on lab exercises.