

LVM, RAID, XFS and EXT3 file systems tuning for small files massive heavy load concurrent parallel I/O on Debian

Author: André Felipe Machado<andremachado@techforce.com.br>

Thousands concurrent parallel read write accesses over tens of millions of small files is a terrible performance tuning problem for e-mail servers.

You must understand and fine tune all your infrastructure chain, following the previous articles for [data storage Debian 5.x](#) and [Lenny](#).

We reduced the CPU I/O wait from 30% to 0,3% (XFS) and 5% (EXT3) with these combined previously undocumented file system tuning tips.

Mainframes have excellent parallel I/O performance. Data storage systems could deliver parallel I/O performance, too. Debian GNU/Linux if you [configure them](#) [multipath connections](#) understand the concepts, application behaviour and follow some hints for configuring and tuning your LVM, RAID, XFS, EXT3.

The key word is

PARALLEL.

Fiber channel data storage systems, multipath connections, excell at parallel I/O.

These days of multicore, multi gigabyte RAM, fiber channel and many channels gigabit connected Debian GNU / Linux systems serving thousands of concurrent users need PARALLEL I/O performance and new approach to file system tuning.

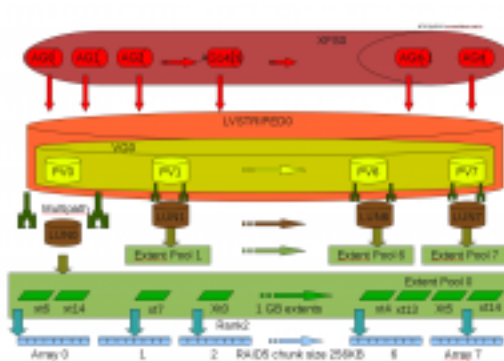
Serial and single threaded file system benchmarks are almost meaningless here. You could [modify application behaviour with the Lenny backported multiprocess fio](#)

You must hunt for lowest CPU I/O wait.

The whole concept graphic:

If you could understand the following concept graphic, you could solve your performance problems by tuning your Debian system for your hardware and application behaviour.

As you realize from the concept graphic, we are accessing 512 Fiber Channel 15 krpm drives concurrently in parallel.



How to configure LVM and or RAID

LVM striping or RAID striping without parity.

It is not needed to configure parity space because the storage block device already execute this.

Calculate the stripe width and size:

Background info:

The storage virtual stripe size (the extents) is 1 GB and the raid chunk size is 256 KB.

The best sw raid performance was achieved with a LVM stripe of 4 MB (the Linux max) and or raid chunk size 256 KB.

Leave the low level details to the data storage device.

Powers of 2 are important

For best performance, you should choose powers of 2 numbers to keep aligned the RAID, LVM, software RAID, and file systems internals like strides and allocation groups.

If something is not aligned, low level calculations will be performed to position each file and performance will degrade.

As the IBM DS 8300 data storage systems we use are internally configured with 8 Fiber Channel disks forming each RAID5 ranks at each extent pool, we choose *eight* as the key number for our configurations.

Example:

```
pvcreate /dev/mapper/mpath0 up to pvcreate /dev/mapper/mpath7
```

Example:

```
# vgcreate --autobackup y grup01 /dev/mapper/mpath0 /dev/mapper/mpath1 \  
/dev/mapper/mpath2 /dev/mapper/mpath3 /dev/mapper/mpath4 \  
/dev/mapper/mpath5 \  
/dev/mapper/mpath6 /dev/mapper/mpath7
```

Exemple:

```
# lvcreate --autobackup y --readahead auto --stripes 8 --stripesize  
4096 \
```

How to choose, format, mount and tune the filesystem

Choose the filesystem suitable for YOUR work load and profile.

- Ext3 does not have "good" parallel I/O.
- XFS has optimal behaviour for parallel I/O threads, if formatted using multiple Allocation Groups.
- Ext3 has better performance for small files at 1 read/write thread.
- XFS has better performance for big files and multiple read/write threads.
- It is difficult to choose and tune when one has small files with multiple threads or big files with 1 thread.
- Ext3 has better performance after formatting, but fragments with time under multiple threads, increasing latency and lowering performance.
- The actual performance observation of OUR SMTP servers workload shows that XFS has MUCH better performance under heavy parallel I/O, both fs tuned. Under the same workload, the XFS does not have more than 0,3% I/O wait. The ext3 has 5% I/O wait under OUR load.

Plan the filesystem formatting by the LVM / RAID stripe size and number.

XFS allows
almost automatic LVM stripe alignment.

Ext3 stripes
MUST be aligned to the LVM/RAID stripes by calculation:

- [Ext3 stride calculator](#).
- [Ext3 stride presentation](#).

Format the filesystem to suit the intended usage:

Ext3:

study the /etc/mke2fs.conf to choose the "-T" parameter.

XFS:

Small files:

Update 28 dec 2010: Allocation Groups size = user quota

Further tests and application behaviour under actual users load demonstrated that setting AG size equal to user quota (or email user quota) gives the best trade-off performance for small files.

For example, for an e-mail user quota of 256 MB, we used:

```
mkfs.xfs -l internal,lazy-count=1,size=128m -d agsize=256m -b  
size=4096 /dev/grupo01/volume01
```

All our email servers are now configured this way and performing very well under parallel load at our environment with a n6040 and an exclusive DS8300.

Old minimum size AG approach:

Maximize the Allocation Group number up to the size of an email user quota.

It is because the way XFS creates directories. XFS **tries** to create new directories at a yet empty allocation group.

At linux 2.6.26, each AG may have a minimum **2²⁷** bytes size, **WITHOUT ROUNDING**, for default block, sectors and inode sizes. The XFS source allows 16 MB, but may be an overkill.

The

EXACT storage size may be obtained from a first format try, collected by `xfs_info`, multiplying the number of blocks by the block size (4096 default).

max AG number = (exact storage size / (2²⁷))

Example: `-l internal,lazy-count=1,size=128m -d agcount=399 -b size=4096`

Example: `-l internal,lazy-count=1,size=128m -d agsize=268435456 -b size=4096`

You "may" install the metadata log (128 MB max) at an external LUN for a bit more performance, but evaluate the trade offs. If you have another extent pool formed by SSD hard drives, you may try to allocate these small LUNs there for the metadata log.

Big files:

One Allocation Group for each processor core.

Mount the filesystem tuned to suit the intended usage:

XFS:

`noatime,nodiratime,attr2,nobarrier,logbufs=8,logbsize=256k,osyncisdsync`

- `nobarrier` only for high end storage.
- `osyncisdsync` not for databases.

Ext3:

`noatime,nodiratime,async,commit=1,data=journal,reservation`

- async not for databases.
- commit=1 for heavy loaded servers.
- data=journal only for heavy loaded servers and only for SOME parallel I/O profiles.
- inode reservation released at kernel 2.6.13.
- inode reservation improves the performance for multithreaded ext3 writes.
But badly fragments over time under multithreaded I/O.

What is ext3 inode reservation?

The ext3 inode reservation pre-allocates some inodes to each directory, to improve performance when a new file should be created.

The filesystem does not need to rescan inode table, looking for enough available space when a new file should be created or appended.

This new still undocumented feature improves ext3 performance a lot.

But with the preallocation, the ext3 fragments badly over time. Specially when reaching full capacity. This ceiling is lower than the XFS.

Why XFS has better parallel I/O behaviour?

EXT3 has one inode table.

XFS has one inode table for each allocation group.

Each allocation group could be modified concurrently, almost without affecting other allocation groups if the file could be written entirely into one allocation group.

When under heavy concurrent parallel I/O, the ext3 single inode table will cause I/O queue contention.