XFS FAQ

From XFS.org

Info from: main XFS faq at SGI (http://oss.sgi.com/projects/xfs/faq.html)

Many thanks to earlier maintainers of this document - Thomas Graichen and Seth Mos.

Contents

Q: Where can I find documentation about XFS?

The SGI XFS project page http://oss.sgi.com/projects/xfs/ is the definitive reference. It contains pointers to whitepapers, books, articles, etc.

You could also join the XFS mailing list or the **#xfs** IRC channel on *irc.freenode.net*.

Q: Where can I find documentation about ACLs?

Andreas Gruenbacher maintains the Extended Attribute and POSIX ACL documentation for Linux at http://acl.bestbits.at/

The **acl(5)** manual page is also quite extensive.

Q: Where can I find information about the internals of XFS?

An SGI XFS Training course (http://oss.sgi.com/projects/xfs/training/) aimed at developers, triage and support staff, and serious users has been in development. Parts of the course are clearly still incomplete, but there is enough content to be useful to a broad range of users.

Barry Naujok has documented the XFS ondisk format (http://oss.sgi.com/projects/xfs/papers/xfs_filesystem_structure.pdf) which is a very useful reference.

Q: What partition type should I use for XFS on Linux?

Linux native filesystem (83).

Q: What mount options does XFS have?

There are a number of mount options influencing XFS filesystems - refer to the **mount(8)** manual page or the documentation in the kernel source tree itself (Documentation/filesystems/xfs.txt (http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/filesystems/xfs.txt;hb=HEAD))

Q: Is there any relation between the XFS utilities and the kernel version?

No, there is no relation. Newer utilities tend to mainly have fixes and checks the previous versions might not have. New features are also added in a backward compatible way - if they are enabled via mkfs, an incapable (old) kernel will recognize that it does not understand the new feature, and refuse to mount the filesystem.

Q: Does it run on platforms other than i386?

XFS runs on all of the platforms that Linux supports. It is more tested on the more common platforms, especially the i386 family. Its also well tested on the IA64 platform since thats the platform SGI Linux products use.

Q: Quota: Do quotas work on XFS?

Yes.

To use quotas with XFS, you need to enable XFS quota support when you configure your kernel. You also need to specify quota support when mounting. You can get the Linux quota utilities at their sourceforge website http://sourceforge.net/projects/linuxquota/ or use **xfs_quota(8)**.

Q: Quota: What's project quota?

The project quota is a quota mechanism in XFS can be used to implement a form of directory tree quota, where a specified directory and all of the files and subdirectories below it (i.e. a tree) can be restricted to using a subset of the available space in the filesystem.

Q: Quota: Can group quota and project quota be used at the same time?

No, project quota cannot be used with group quota at the same time. On the other hand user quota and project quota can be used simultaneously.

Q: Quota: Is umounting prjquota (project quota) enabled fs and mounting it again with grpquota (group quota) removing prjquota limits previously set from fs (and vice versa) ?

To be answered.

Q: Are there any dump/restore tools for XFS?

xfsdump(8) and **xfsrestore(8)** are fully supported. The tape format is the same as on IRIX, so tapes are interchangeable between operating systems.

Q: Does LILO work with XFS?

This depends on where you install LILO.

Yes, for MBR (Master Boot Record) installations.

No, for root partition installations because the XFS superblock is written at block zero, where LILO would be installed. This is to maintain compatibility with the IRIX on-disk format, and will not be changed.

Q: Does GRUB work with XFS?

There is native XFS filesystem support for GRUB starting with version 0.91 and onward. Unfortunately, GRUB used to make incorrect assumptions about being able to read a block device image while a filesystem is mounted and actively being written to, which could cause intermittent problems when using XFS. This has reportedly since been fixed, and the 0.97 version (at least) of GRUB is apparently stable.

Q: Can XFS be used for a root filesystem?

Yes, with one caveat: Linux does not support an external XFS journal for the root filesystem via the "rootflags=" kernel parameter. To use an external journal for the root filesystem in Linux, an init ramdisk must mount the root filesystem with explicit "logdev=" specified. More information here. (http://mindplusplus.wordpress.com/2008/07/27/scratching-an-i.html)

Q: Will I be able to use my IRIX XFS filesystems on Linux?

Yes. The on-disk format of XFS is the same on IRIX and Linux. Obviously, you should back-up your data before trying to move it between systems. Filesystems must be "clean" when moved (i.e. unmounted). If you plan to use IRIX filesystems on Linux keep the following points in mind: the kernel needs to have SGI partition support enabled; there is no XLV support in Linux, so you are unable to read IRIX filesystems which use the XLV volume manager; also not all blocksizes available on IRIX are available on Linux (only blocksizes less than or equal to the pagesize of the architecture: 4k for i386, ppc, ... 8k for alpha, sparc, ... is possible for now). Make sure that the directory format is version 2 on the IRIX filesystems (this is the default since IRIX 6.5.5). Linux can only read v2 directories.

Q: Is there a way to make a XFS filesystem larger or smaller?

You can *NOT* make a XFS partition smaller online. The only way to shrink is to do a complete dump, mkfs and restore.

An XFS filesystem may be enlarged by using **xfs_growfs(8)**.

If using partitions, you need to have free space after this partition to do so. Remove partition, recreate it larger with the *exact same* starting point. Run **xfs_growfs** to make the partition larger. Note - editing partition tables is a dangerous pastime, so back up your filesystem before doing so.

Using XFS filesystems on top of a volume manager makes this a lot easier.

Q: What information should I include when reporting a problem?

What you need to report depend on the problem you are seeing. Firstly, your machine hardware and storage configuration needs to be described. That includes:

- kernel version (uname -a)
- xfsprogs version (xfs_repair -V)
- number of CPUs
- contents of /proc/meminfo
- contents of /proc/mounts
- contents of /proc/partitions
- RAID layout (hardware and/or software)
- LVM configuration
- type of disks you are using
- write cache status of drives
- size of BBWC and mode it is running in
- xfs_info output on the filesystem in question
- dmesg output showing all error messages and stack traces

Then you need to describe your workload that is causing the problem, and a demonstration of the bad behaviour that is occurring. If it is a performance problem, then 30s - 1 minute samples of:

1. iostat -x -d -m 5
2. vmstat 5

can give us insight into the IO and memory utilisation of your machine at the time of the problem.

If the filesystem is hanging, then capture the output of the dmesg command after running:

```
# echo w > /proc/sysrq-trigger
# dmesg
```

will tell us all the hung processes in the machine, often pointing us directly to the cause of the hang.

And for advanced users, capturing an event trace using **trace-cmd** (git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git) will be very helpful. In many cases the XFS developers will ask for this information anyway, so it's a good idea to be ready with it in advance. Start the trace with this command, either from a directory not on an XFS filesystem or with an output file destination on a non-XFS filesystem:

```
# trace-cmd record -e xfs\*
```

before the problem occurs, and once it has occurred, kill the trace-cmd with ctrl-C, and then run:

```
# trace-cmd report > trace_report.txt
```

Compress the trace_report.txt file and include that with the bug report. The reason for trying to host the output of the record command on a different filesystem is so that the writing of the output file does not pollute the trace of the problem we are trying to diagnose.

If you have a problem with **xfs_repair(8)**, make sure that you save the entire output of the problematic run so that the developers can see exactly where it encountered the problem. You might be asked to capture the metadata in the filesystem using **xfs_metadump(8)** (which obfuscates filenames and attributes to protect your privacy) and make the dump available for someone to analyse.

Q: Mounting an XFS filesystem does not work - what is wrong?

If mount prints an error message something like:

```
   mount: /dev/hda5 has wrong major or minor number
```

you either do not have XFS compiled into the kernel (or you forgot to load the modules) or you did not use the "-t xfs" option on mount or the "xfs" option in `/etc/fstab`.

If you get something like:

```
mount: wrong fs type, bad option, bad superblock on /dev/sda1,
       or too many mounted file systems
```

Refer to your system log file (`/var/log/messages`) for a detailed diagnostic message from the kernel.

Q: Does the filesystem have an undelete capability?

There is no undelete in XFS.

However, if an inode is unlinked but neither it nor its associated data blocks get immediately re-used and overwritten, there is some small chance to recover the file from the disk.

photorec, *xfs_irecover* or *xfsr* are some tools which attempt to do this, with varying success.

There are also commercial data recovery services and closed source software like Raise Data Recovery for XFS (http://www.ufsexplorer.com/rdr_xfs.php) which claims to recover data, although this has not been tested by the XFS developers.

As always, the best advice is to keep good backups.

Q: How can I backup a XFS filesystem and ACLs?

You can backup a XFS filesystem with utilities like **xfsdump(8)** and standard **tar(1)** for standard files. If you want to backup ACLs you will need to use **xfsdump** or Bacula (http://www.bacula.org/en/dev-manual/Current_State_Bacula.html) (> version 3.1.4) or rsync (http://rsync.samba.org/) (>= version 3.0.0) to backup ACLs and EAs. **xfsdump** can also be integrated with amanda(8) (http://www.amanda.org/) .

Q: I see applications returning error 990 or "Structure needs cleaning", what is wrong?

The error 990 stands for EFSCORRUPTED (http://oss.sgi.com/cgi-bin/gitweb.cgi?p=xfs/xfs.git;a=blob;f=fs/xfs/linux-2.6/xfs_linux.h#l145) which usually means XFS has detected a filesystem metadata problem and has shut the filesystem down to prevent further damage. Also, since about June 2006, we converted from EFSCORRUPTED/990 over to using EUCLEAN (http://oss.sgi.com/cgi-bin/gitweb.cgi?p=xfs/xfs.git;a=commit;h=da2f4d679c8070ba5b6a920281e495917b293aa0) , "Structure needs cleaning."

The cause can be pretty much anything, unfortunately - filesystem, virtual memory manager, volume manager, device driver, or hardware.

There should be a detailed console message when this initially happens. The messages have important information giving hints to developers as to the earliest point that a problem was detected. It is there to protect your data.

You can use xfs_repair to remedy the problem (with the file system unmounted).

Q: Why do I see binary NULLS in some files after recovery when I unplugged the power?

Update: This issue has been addressed with a CVS fix on the 29th March 2007 and merged into mainline on 8th May 2007 for 2.6.22-rc1.

XFS journals metadata updates, not data updates. After a crash you are supposed to get a consistent filesystem which looks like the state sometime shortly before the crash, NOT what the in memory image looked like the instant before the crash.

Since XFS does not write data out immediately unless you tell it to with fsync, an O_SYNC or O_DIRECT open (the same is true of other filesystems), you are looking at an inode which was flushed out, but whose data was not. Typically you'll find that the inode is not taking any space since all it has is a size but no extents allocated (try examining the file with the **xfs_bmap(8)** command).

Q: What is the problem with the write cache on journaled filesystems?

Many drives use a write back cache in order to speed up the performance of writes. However, there are conditions such as power failure when the write cache memory is never flushed to the actual disk. Further, the drive can de-stage data from the write cache to the platters in any order that it chooses. This causes problems for XFS and journaled filesystems in general because they rely on knowing when a write has completed to the disk. They need to know that the log information has made it to disk before allowing metadata to go to disk. When the metadata makes it to disk then the transaction can effectively be deleted from the log resulting in movement of the tail of the log and thus freeing up some log space. So if the writes never make it to the physical disk, then the ordering is violated and the log and metadata can be lost, resulting in filesystem corruption.

With hard disk cache sizes of currently (Jan 2009) up to 32MB that can be a lot of valuable information. In a RAID with 8 such disks these adds to 256MB, and the chance of having filesystem metadata in the cache is so high that you have a very high chance of big data losses on a power outage.

With a single hard disk and barriers turned on (on=default), the drive write cache is flushed before and after a barrier is issued. A powerfail "only" loses data in the cache but no essential ordering is violated, and corruption will not occur.

With a RAID controller with battery backed controller cache and cache in write back mode, you should turn off barriers - they are unnecessary in this case, and if the controller honors the cache flushes, it will be harmful to performance. But then you *must* disable the individual hard disk write cache in order to ensure to keep the filesystem intact after a power failure. The method for doing this is different for each RAID controller. See the section about RAID controllers below.

Q: How can I tell if I have the disk write cache enabled?

For SCSI/SATA:

- Look in dmesg(8) output for a driver line, such as:
  "SCSI device sda: drive cache: write back"
- # sginfo -c /dev/sda | grep -i 'write cache'

For PATA/SATA (for SATA this requires at least kernel 2.6.15 because ATA command passthrough support (http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=b095518ef51c37658c58367bd19240b8a113f25c)):

- # hdparm -I /dev/sda
  and look under "Enabled Supported" for "Write cache"

For RAID controllers:

- See the section about RAID controllers below

Q: How can I address the problem with the disk write cache?

Disabling the disk write back cache.

For SATA/PATA(IDE) (for SATA this requires at least kernel 2.6.15 because ATA command passthrough support (http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=b095518ef51c37658c58367bd19240b8a113f25c)):

- # hdparm -W0 /dev/sda
  # hdparm -W0 /dev/hda
- # blktool /dev/sda wcache off
  # blktool /dev/hda wcache off

For SCSI:

- Using sginfo(8) which is a little tedious
  It takes 3 steps. For example:
  1. #sginfo -c /dev/sda

which gives a list of attribute names and values
2. #sginfo -cX /dev/sda
which gives an array of cache values which you must match up with from step 1, e.g.
0 0 0 1 0 1 0 0 0 0 65535 0 65535 65535 1 0 0 0 3 0 0
3. #sginfo -cXR /dev/sda 0 0 0 1 0 0 0 0 0 0 65535 0 65535 65535 1 0 0 0 3 0 0
allows you to reset the value of the cache attributes.

For RAID controllers:

- See the section about RAID controllers below

This disabling is kept persistent for a SCSI disk. However, for a SATA/PATA disk this needs to be done after every reset as it will reset back to the default of the write cache enabled. And a reset can happen after reboot or on error recovery of the drive. This makes it rather difficult to guarantee that the write cache is maintained as disabled.

Using an external log.

Some people have considered the idea of using an external log on a separate drive with the write cache disabled and the rest of the file system on another disk with the write cache enabled. However, that will **not** solve the problem. For example, the tail of the log is moved when we are notified that a metadata write is completed to disk and we won't be able to guarantee that if the metadata is on a drive with the write cache enabled.

In fact using an external log will disable XFS' write barrier support.

Write barrier support.

Write barrier support is enabled by default in XFS since kernel version 2.6.17. It is disabled by mounting the filesystem with "nobarrier". Barrier support will flush the write back cache at the appropriate times (such as on XFS log writes). This is generally the recommended solution, however, you should check the system logs to ensure it was successful. Barriers will be disabled and reported in the log if any of the 3 scenarios occurs:

- "Disabling barriers, not supported with external log device"
- "Disabling barriers, not supported by the underlying device"
- "Disabling barriers, trial barrier write failed"

If the filesystem is mounted with an external log device then we currently don't support flushing to the data and log devices (this may change in the future). If the driver tells the block layer that the device does not support write cache flushing with the write cache enabled then it will report that the device doesn't support it. And finally we will actually test out a barrier write on the superblock and test its error state afterwards, reporting if it fails.

Q. Should barriers be enabled with storage which has a persistent write cache?

Many hardware RAIDs have a persistent write cache which is preserved across power failure, interface resets, system crashes, etc. The same may be true of some SSD devices. This sort of hardware should report to the operating system that no flushes are required, and in that case barriers will not be issued, even without the "nobarrier" option. Quoting Christoph Hellwig on the xfs list (http://oss.sgi.com/archives/xfs/2015-12/msg00281.html) ,

```
If the device does not need cache flushes it should not report requiring
flushes, in which case nobarrier will be a noop.  Or to phrase it
differently:  If nobarrier makes a difference skipping it is not safe.
```

On modern kernels with hardware which properly reports write cache behavior, there is no need to change barrier options at mount time.

Q. Which settings does my RAID controller need ?

It's hard to tell because there are so many controllers. Please consult your RAID controller documentation to determine how to change these settings, but we try to give an overview here:

Real RAID controllers (not those found onboard of mainboards) normally have a battery or flash backed cache (or an ultracapacitor (http://en.wikipedia.org/wiki/Electric_double-layer_capacitor) + flash memory "zero maintenance cache (http://www.tweaktown.com/articles/2800/adaptec_zero_maintenance_cache_protection_explained/) ") which is used for buffering writes to improve speed. This battery backed cache should ensure that if power fails or a PSU dies, the contents of the cache will be written to disk on next boot. However, the individual hard disk write caches need to be turned off, as they are not protected from a powerfail and will just lose all contents.

If you do not have a battery or flash backed cache you should seriously consider disabling write cache if you value your data.

- onboard RAID controllers: there are so many different types it's hard to tell. Generally, those controllers have no cache, but let the hard disk write cache on. That can lead to the bad situation that after a powerfail with RAID-1 when only parts of the disk cache have been written, the controller doesn't even see that the disks are out of sync, as the disks can resort cached blocks and might have saved the superblock info, but then lost different data contents. So, turn off disk write caches before using the RAID function.
- 3ware: /cX/uX set cache=off, this will disable the controller and disk cache

(see http://www.3ware.com/support/UserDocs/CLIGuide-9.5.1.1.pdf, page 86);

- Adaptec: allows setting individual drives cache

arcconf setcache <disk> wb|wt wb=write back, which means write cache on, wt=write through, which means write cache off. So "wt" should be chosen.

- Areca: In archttp under "System Controls" -> "System Configuration" there's the option "Disk Write Cache Mode" (defaults "Auto")

"Off": disk write cache is turned off

"On": disk write cache is enabled, this is not safe for your data but fast

"Auto": If you use a BBM (battery backup module, which you really should use if you care about your data), the controller automatically turns disk writes off, to protect your data. In case no BBM is attached, the controller switches to "On", because neither controller cache nor disk cache is safe so you don't seem to care about your data and just want high speed (which you get then).

That's a very sensible default so you can let it "Auto" or enforce "Off" to be sure.

- LSI MegaRAID: allows setting individual disks cache:

```
MegaCli -AdpCacheFlush -aN|-a0,1,2|-aALL                          # flushes the controller cache
MegaCli -LDGetProp -Cache    -LN|-L0,1,2|-LAll -aN|-a0,1,2|-aALL  # shows the controller cache settin
MegaCli -LDGetProp -DskCache -LN|-L0,1,2|-LAll -aN|-a0,1,2|-aALL  # shows the disk cache settings (fo
MegaCli -LDSetProp -EnDskCache|DisDskCache  -LN|-L0,1,2|-LAll  -aN|-a0,1,2|-aALL # set disk cache set
```

- Xyratex: from the docs: "Write cache includes the disk drive cache and controller cache.". So that means you can only set the drive caches and the unit caches together. To protect your data, turn it off, but write performance will suffer badly as also the controller write cache is disabled.

Q: Which settings are best with virtualization like VMware, XEN, qemu?

The biggest problem is that those products seem to also virtualize disk writes in a way that even barriers don't work any more, which means even a fsync is not reliable. Tests confirm that unplugging the power from such a system even with RAID controller with battery backed cache and hard disk cache turned off (which is safe on a normal host) you can destroy a database within the virtual machine (client, domU whatever you call it).

In qemu you can specify cache=off on the line specifying the virtual disk. For others information is missing.

Q: What is the issue with directory corruption in Linux 2.6.17?

In the Linux kernel 2.6.17 release a subtle bug was accidentally introduced into the XFS directory code by some "sparse" endian annotations. This bug was sufficiently uncommon (it only affects a certain type of format change, in Node or B-Tree format directories, and only in certain situations) that it was not detected during our regular regression testing, but it has been observed in the wild by a number of people now.

Update: the fix is included in 2.6.17.7 and later kernels.

To add insult to injury, **xfs_repair(8)** is currently not correcting these directories on detection of this corrupt state either. This **xfs_repair** issue is actively being worked on, and a fixed version will be available shortly.

*Update: a fixed **xfs_repair** is now available; version 2.8.10 or later of the xfsprogs package contains the fixed version.*

No other kernel versions are affected. However, using a corrupt filesystem on other kernels can still result in the filesystem being shutdown if the problem has not been rectified (on disk), making it seem like other kernels are affected.

xfs_repair -n should be able to detect any directory corruption.

Until a fixed **xfs_repair** binary is available, one can make use of the **xfs_db(8)** command to mark the problem directory for removal (see the example below). A subsequent **xfs_repair** invocation will remove the directory and move all contents into "lost+found", named by inode number (see second example on how to map inode number to directory entry name, which needs to be done _before_ removing the directory itself). The inode number of the corrupt directory is included in the shutdown report issued by the kernel on detection of directory corruption. Using that inode number, this is how one would ensure it is removed:

```
# xfs_db -x /dev/sdXXX
xfs_db> inode NNN
xfs_db> print
core.magic = 0x494e
core.mode = 040755
core.version = 2
core.format = 3 (btree)
...
xfs_db> write core.mode 0
xfs_db> quit
```

A subsequent **xfs_repair** will clear the directory, and add new entries (named by inode number) in lost+found.

The easiest way to map inode numbers to full paths is via **xfs_ncheck(8)**:

```
# xfs_ncheck -i 14101 -i 14102 /dev/sdXXX
      14101 full/path/mumble_fratz_foo_bar_1495
      14102 full/path/mumble_fratz_foo_bar_1494
```

Should this not work, we can manually map inode numbers in B-Tree format directory by taking the following steps:

```
# xfs_db -x /dev/sdXXX
xfs_db> inode NNN
xfs_db> print
core.magic = 0x494e
...
next_unlinked = null
u.bmbt.level = 1
u.bmbt.numrecs = 1
u.bmbt.keys[1] = [startoff] 1:[0]
u.bmbt.ptrs[1] = 1:3628
xfs_db> fsblock 3628
xfs_db> type bmapbtd
xfs_db> print
magic = 0x424d4150
level = 0
numrecs = 19
leftsib = null
rightsib = null
recs[1-19] = [startoff,startblock,blockcount,extentflag]
       1:[0,3088,4,0] 2:[4,3128,8,0] 3:[12,3308,4,0] 4:[16,3360,4,0]
       5:[20,3496,8,0] 6:[28,3552,8,0] 7:[36,3624,4,0] 8:[40,3633,4,0]
       9:[44,3688,8,0] 10:[52,3744,4,0] 11:[56,3784,8,0]
       12:[64,3840,8,0] 13:[72,3896,4,0] 14:[33554432,3092,4,0]
       15:[33554436,3488,8,0] 16:[33554444,3629,4,0]
       17:[33554448,3748,4,0] 18:[33554452,3900,4,0]
       19:[67108864,3364,4,0]
```

At this point we are looking at the extents that hold all of the directory information. There are three types of extent here, we have the data blocks (extents 1 through 13 above), then the leaf blocks (extents 14 through 18), then the freelist blocks (extent 19 above). The jumps in the first field (start offset) indicate our progression through each of the three types. For recovering file names, we are only interested in the data blocks, so we can now feed those offset numbers into the **xfs_db** dblock command. So, for the fifth extent - 5:[20,3496,8,0] - listed above:

```
...
xfs_db> dblock 20
xfs_db> print
dhdr.magic = 0x58443244
dhdr.bestfree[0].offset = 0
dhdr.bestfree[0].length = 0
dhdr.bestfree[1].offset = 0
dhdr.bestfree[1].length = 0
dhdr.bestfree[2].offset = 0
dhdr.bestfree[2].length = 0
du[0].inumber = 13937
du[0].namelen = 25
```

```
du[0].name = "mumble_fratz_foo_bar_1595"
du[0].tag = 0x10
du[1].inumber = 13938
du[1].namelen = 25
du[1].name = "mumble_fratz_foo_bar_1594"
du[1].tag = 0x38
...
```

So, here we can see that inode number 13938 matches up with name "mumble_fratz_foo_bar_1594". Iterate through all the extents, and extract all the name-to-inode-number mappings you can, as these will be useful when looking at "lost+found" (once **xfs_repair** has removed the corrupt directory).

Q: Why does my > 2TB XFS partition disappear when I reboot ?

Strictly speaking this is not an XFS problem.

To support > 2TB partitions you need two things: a kernel that supports large block devices (`CONFIG_LBD=y`) and a partition table format that can hold large partitions. The default DOS partition tables don't. The best partition format for > 2TB partitions is the EFI GPT format (`CONFIG_EFI_PARTITION=y`).

Without CONFIG_LBD=y you can't even create the filesystem, but without `CONFIG_EFI_PARTITION=y` it works fine until you reboot at which point the partition will disappear. Note that you need to enable the `CONFIG_PARTITION_ADVANCED` option before you can set `CONFIG_EFI_PARTITION=y`.

Q: Why do I receive `No space left on device` after `xfs_growfs`?

After growing a XFS filesystem (http://oss.sgi.com/archives/xfs/2009-01/msg01023.html) , df(1) would show enough free space but attempts to write to the filesystem result in -ENOSPC. This was an issue with the older "inode32" inode allocation mode, where inode allocation is restricted to lower filesysetm blocks. To fix this, Dave Chinner advised (http://oss.sgi.com/archives/xfs/2009-01/msg01031.html) :

```
The only way to fix this is to move data around to free up space
below 1TB. Find your oldest data (i.e. that was around before even
the first grow) and move it off the filesystem (move, not copy).
Then if you copy it back on, the data blocks will end up above 1TB
and that should leave you with plenty of space for inodes below 1TB.

A complete dump and restore will also fix the problem ;)
```

Alternately, you can add 'inode64' to your mount options to allow inodes to live above 1TB.

example:No space left on device on xfs filesystem with 7.7TB free (https://www.centos.org/modules/newbb/viewtopic.php?topic_id=30703&forum=38)

However, 'inode64' has been the default behavior since kernel v3.7...

Unfortunately, v3.7 also added a bug present from kernel v3.7 to v3.17 which caused new allocation groups added by growfs to be unavailable for inode allocation. This was fixed by commit `9de67c3b xfs: allow inode allocations in post-growfs disk space. (http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=9de67c3ba9ea961ba420573d56479d09d33a7587)` in kernel v3.17. Without that commit, the problem can be worked around by doing a "mount -o remount,inode64" after the growfs operation.

Q: Is using noatime or/and nodiratime at mount time giving any performance benefits in xfs (or not using them performance decrease)?

The default atime behaviour is relatime, which has almost no overhead compared to noatime but still maintains sane atime values. All Linux filesystems use this as the default now (since around 2.6.30), but XFS has used relatime-like behaviour since 2006, so no-one should really need to ever use noatime on XFS for performance reasons.

Also, noatime implies nodiratime, so there is never a need to specify nodiratime when noatime is also specified.

Q: How to get around a bad inode repair is unable to clean up

The trick is go in with xfs_db and mark the inode as a deleted, which will cause repair to clean it up and finish the remove process.

```
xfs_db -x -c 'inode XXX' -c 'write core.nextents 0' -c 'write core.size 0' /dev/hdXX
```

Q: How to calculate the correct sunit,swidth values for optimal performance

XFS allows to optimize for a given RAID stripe unit (stripe size) and stripe width (number of data disks) via mount options.

These options can be sometimes autodetected (for example with md raid and recent enough kernel (>= 2.6.32) and xfsprogs (>= 3.1.1) built with libblkid support) but manual calculation is needed for most of hardware raids.

The calculation of these values is quite simple:

```
su = <RAID controllers stripe size in BYTES (or KiBytes when used with k)>
sw = <# of data disks (don't count parity disks)>
```

So if your RAID controller has a stripe size of 64KB, and you have a RAID-6 with 8 disks, use

```
su = 64k
sw = 6 (RAID-6 of 8 disks has 6 data disks)
```

A RAID stripe size of 256KB with a RAID-10 over 16 disks should use

```
su = 256k
sw = 8 (RAID-10 of 16 disks has 8 data disks)
```

Alternatively, you can use "sunit" instead of "su" and "swidth" instead of "sw" but then sunit/swidth values need to be specified in "number of 512B sectors"!

Note that `xfs_info` and `mkfs.xfs` interpret sunit and swidth as being specified in units of 512B sectors; that's unfortunately not the unit they're reported in, however. `xfs_info` and `mkfs.xfs` report them in multiples of your basic block size (bsize) and not in 512B sectors.

Assume for example: swidth 1024 (specified at mkfs.xfs command line; so 1024 of 512B sectors) and block size of 4096 (bsize reported by mkfs.xfs at output). You should see swidth 128 (reported by mkfs.xfs at output). 128 * 4096 == 1024 * 512.

When creating XFS filesystem on top of LVM on top of hardware raid please use sunit/swith values as when creating XFS filesystem directly on top of hardware raid.

Q: Why doesn't NFS-exporting subdirectories of inode64-mounted filesystem work?

The default `fsid` type encodes only 32-bit of the inode number for subdirectory exports. However, exporting the root of the filesystem works, or using one of the non-default `fsid` types (`fsid=uuid` in `/etc/exports` with recent `nfs-utils`) should work as well. (Thanks, Christoph!)

Q: What is the inode64 mount option for?

By default, with 32bit inodes, XFS places inodes only in the first 1TB of a disk. If you have a disk with 100TB, all inodes will be stuck in the first TB. This can lead to strange things like "disk full" when you still have plenty space free, but there's no more place in the first TB to create a new inode. Also, performance sucks.

To come around this, use the inode64 mount options for filesystems >1TB. Inodes will then be placed in the location where their data is, minimizing disk seeks.

Beware that some old programs might have problems reading 64bit inodes, especially over NFS. Your editor used inode64 for over a year with recent (openSUSE 11.1 and higher) distributions using NFS and Samba without any corruptions, so that might be a recent enough distro.

Q: Can I just try the inode64 option to see if it helps me?

Starting from kernel 2.6.35, you can try and then switch back. Older kernels have a bug leading to strange problems if you mount without inode64 again. For example, you can't access files & dirs that have been created with an inode >32bit anymore.

Q: Performance: mkfs.xfs -n size=64k option

Asking the implications of that mkfs option on the XFS mailing list, Dave Chinner explained it this way:

Inodes are not stored in the directory structure, only the directory entry name and the inode number. Hence the amount of space used by a directory entry is determined by the length of the name.

There is extra overhead to allocate large directory blocks (16 pages instead of one, to begin with, then there's the vmap overhead, etc), so for small directories smaller block sizes are faster for create and unlink operations.

For empty directories, operations on 4k block sized directories consume roughly 50% less CPU that 64k block size directories. The 4k block size directories consume less CPU out to roughly 1.5 million entries where the two are roughly equal. At directory sizes of 10 million entries, 64k directory block operations are consuming about 15% of the CPU that 4k directory block operations consume.

In terms of lookups, the 64k block directory will take less IO but consume more CPU for a given lookup. Hence it depends on your IO latency and whether directory readahead can hide that latency as to which will be faster. e.g. For SSDs, CPU usage might be the limiting factor, not the IO. Right now I don't have any numbers on what the difference might be - I'm getting 1 billion inode population issues worked out first before I start on measuring cold cache lookup times on 1 billion files....

Q: I want to tune my XFS filesystems for <something>

Premature optimization is the root of all evil. - Donald Knuth

The standard answer you will get to this question is this: use the defaults.

There are few workloads where using non-default mkfs.xfs or mount options make much sense. In general, the default values already used are optimised for best performance in the first place. mkfs.xfs will detect the difference between single disk and MD/DM RAID setups and change the default values it uses to configure the filesystem appropriately.

There are a lot of "XFS tuning guides" that Google will find for you - most are old, out of date and full of misleading or just plain incorrect information. Don't expect that tuning your filesystem for optimal bonnie++ numbers will mean your workload will go faster. You should only consider changing the defaults if either: a) you know from experience that your workload causes XFS a specific problem that can be worked around via a configuration change, or b) your workload is demonstrating bad performance when using the default configurations. In this case, you need to understand why your application is causing bad performance before you start tweaking XFS configurations.

In most cases, the only thing you need to to consider for `mkfs.xfs` is specifying the stripe unit and width for hardware RAID devices. For mount options, the only thing that will change metadata performance considerably are the `logbsize` and `delaylog` mount options. Increasing `logbsize` reduces the number of journal IOs for a given workload, and `delaylog` will reduce them even further. The trade off for this increase in metadata performance is that more operations may be "missing" after recovery if the system crashes while actively making modifications.

As of kernel 3.2.12, the default i/o scheduler, CFQ, will defeat much of the parallelization in XFS.

Q: Which factors influence the memory usage of xfs_repair?

This is best explained with an example. The example filesystem is 16Tb, but basically empty (look at icount).

```
# xfs_repair -n -vv -m 1 /dev/vda
Phase 1 - find and verify superblock...
        - max_mem = 1024, icount = 64, imem = 0, dblock = 4294967296, dmem = 2097152
Required memory for repair is greater that the maximum specified
with the -m option. Please increase it to at least 2096.
#
```

xfs_repair is saying it needs at least 2096MB of RAM to repair the filesystem, of which 2,097,152KB is needed for tracking free space. (The -m 1 argument was telling xfs_repair to use ony 1 MB of memory.)

Now if we add some inodes (50 million) to the filesystem (look at icount again), and the result is:

```
# xfs_repair -vv -m 1 /dev/vda
Phase 1 - find and verify superblock...
        - max_mem = 1024, icount = 50401792, imem = 196882, dblock = 4294967296, dmem = 2097152
Required memory for repair is greater that the maximum specified
with the -m option. Please increase it to at least 2289.
```

That is now needs at least another 200MB of RAM to run.

The numbers reported by xfs_repair are the absolute minimum required and approximate at that; more RAM than this may be required to complete successfully. Also, if you only give xfs_repair the minimum required RAM, it will be slow; for best repair performance, the more RAM you can give it the better.

Q: Why some files of my filesystem shows as "?????????? ? ? ? ? ? filename" ?

If ls -l shows you a listing as

```
# ?????????? ? ?      ?          ?                ? file1
  ?????????? ? ?      ?          ?                ? file2
  ?????????? ? ?      ?          ?                ? file3
  ?????????? ? ?      ?          ?                ? file4
```

and errors like:

```
# ls /pathtodir/
  ls: cannot access /pathtodir/file1: Invalid argument
  ls: cannot access /pathtodir/file2: Invalid argument
  ls: cannot access /pathtodir/file3: Invalid argument
  ls: cannot access /pathtodir/file4: Invalid argument
```

or even:

```
# failed to stat /pathtodir/file1
```

It is very probable your filesystem must be mounted with inode64

```
# mount -oremount,inode64 /dev/diskpart /mnt/xfs
```

should make it work ok again. If it works, add the option to fstab.

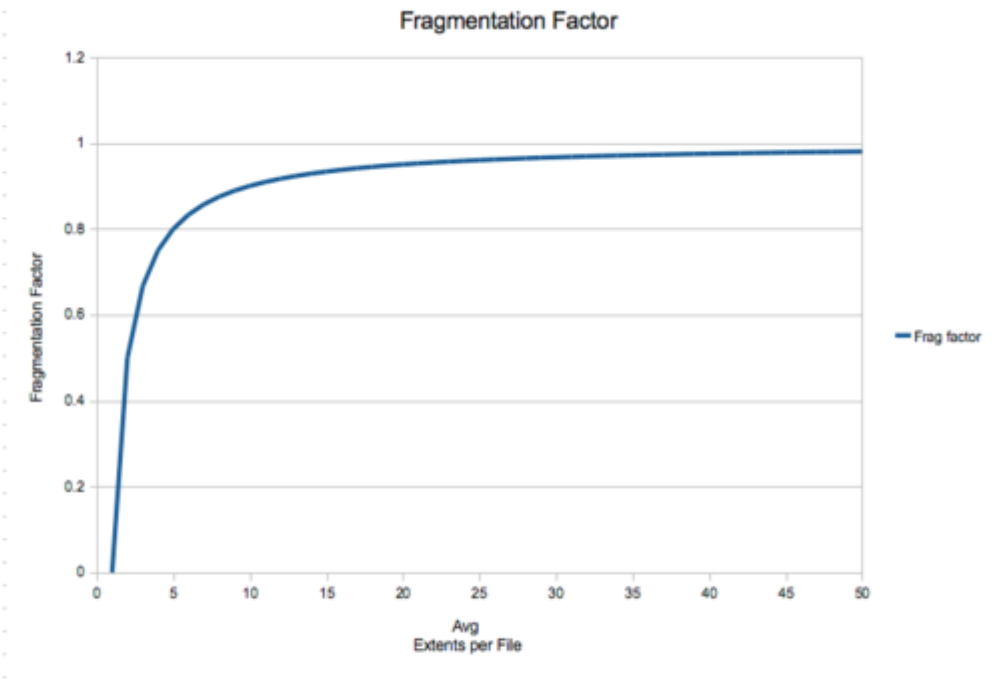
Q: The xfs_db "frag" command says I'm over 50%. Is that bad?

It depends. It's important to know how the value is calculated. xfs_db looks at the extents in all files, and returns:

```
(actual extents - ideal extents) / actual extents
```

This means that if, for example, you have an average of 2 extents per file, you'll get an answer of 50%. 4 extents per file would give you 75%. This may or may not be a problem, especially depending on the size of the files in question. (i.e. 400GB files in four 100GB extents would hardly be considered badly fragmented). The xfs_bmap command can be useful for displaying the actual fragmentation/layout of individual files.

Note that above a few average extents per file, the fragmentation factor rapidly approaches 100%:



Q: I'm getting "Internal error xfs_sb_read_verify" errors when I try to run xfs_growfs under kernels v3.10 through v3.12

This may happen when running xfs_growfs under a v3.10-v3.12 kernel, if the filesystem was previously grown under a kernel prior to v3.8.

Old kernel versions prior to v3.8 did not zero the empty part of new secondary superblocks when growing the filesystem with xfs_growfs.

Kernels v3.10 and later began detecting this non-zero part of the superblock as corruption, and emit the

```
Internal error xfs_sb_read_verify
```

error message.

Kernels v3.13 and later are more forgiving about this - if the non-zero data is found on a Version 4 superblock, it will not be flagged as corruption.

The problematic secondary superblocks may be repaired by using an xfs_repair version 3.2.0-alpha1 or above.

The relevant kernelspace commits are as follows:

```
v3.8  1375cb6 xfs: growfs: don't read garbage for new secondary superblocks <- fixed underlying pr
v3.10 04a1e6c xfs: add CRC checks to the superblock <- detected old underlying problem
v3.13 10e6e65 xfs: be more forgiving of a v4 secondary sb w/ junk in v5 fields <- is more forgivin
```

This commit allows xfs_repair to detect and correct the problem:

```
v3.2.0-alpha1 cbd7508 xfs_repair: zero out unused parts of superblocks
```

Q: Why do files on XFS use more data blocks than expected?

The XFS speculative preallocation algorithm allocates extra blocks beyond end of file (EOF) to minimize file fragmentation during buffered write workloads. Workloads that benefit from this behaviour include slowly growing files, concurrent writers and mixed reader/writer workloads. It also provides fragmentation resistance in situations where memory pressure prevents adequate buffering of dirty data to allow formation of large contiguous regions of data in memory.

This post-EOF block allocation is accounted identically to blocks within EOF. It is visible in 'st_blocks' counts via stat() system calls, accounted as globally allocated space and against quotas that apply to the associated file. The space is reported by various userspace utilities (stat, du, df, ls) and thus provides a common source of confusion for administrators. Post-EOF blocks are temporary in most situations and are usually reclaimed via several possible mechanisms in XFS.

See the FAQ entry on speculative preallocation for details.

Q: What is speculative preallocation?

XFS speculatively preallocates post-EOF blocks on file extending writes in anticipation of future extending writes. The size of a preallocation is dynamic and depends on the runtime state of the file and fs. Generally speaking, preallocation is disabled for very small files and preallocation sizes grow as files grow larger.

Preallocations are capped to the maximum extent size supported by the filesystem. Preallocation size is throttled automatically as the filesystem approaches low free space conditions or other allocation limits on a file (such as a quota).

In most cases, speculative preallocation is automatically reclaimed when a file is closed. Applications that repeatedly trigger preallocation and reclaim cycles (e.g., this is common in file server or log file workloads) can cause fragmentation. Therefore, this pattern is detected and causes the preallocation to persist beyond the lifecycle of the file descriptor.

Q: How can I speed up or avoid delayed removal of speculative preallocation?

Linux 3.8 (and later) includes a scanner to perform background trimming of files with lingering post-EOF preallocations. The scanner bypasses dirty files to avoid interference with ongoing writes. A 5 minute scan interval is used by default and can be adjusted via the following file (value in seconds):

```
     /proc/sys/fs/xfs/speculative_prealloc_lifetime
```

Q: Is speculative preallocation permanent?

Preallocated blocks are normally reclaimed on file close, inode reclaim, unmount or in the background once file write activity subsides. They can be explicitly made permanent via fallocate or a similar interface. They can be implicitly made permanent in situations where file size is extended beyond a range of post-EOF blocks (i.e., via an extending truncate) or following a crash. In the event of a crash, the in-memory state used to track and reclaim the speculative preallocation is lost.

Q: My workload has known characteristics - can I disable speculative preallocation or tune it to an optimal fixed size?

Speculative preallocation can not be disabled but XFS can be tuned to a fixed allocation size with the 'allocsize=' mount option. Speculative preallocation is not dynamically resized when the allocsize mount option is set and thus the potential for fragmentation is increased. Use 'allocsize=64k' to revert to the default XFS behavior prior to support for dynamic speculative preallocation.

Q: mount (or umount) takes minutes or even hours - what could be the reason ?

In some cases xfs log (journal) can become quite big. For example if it accumulates many entries and doesn't get chance to apply these to disk (due to lockup, crash, hard reset etc). xfs will try to reapply these at mount (in dmesg: "Starting recovery (logdev: internal)").

That process with big log to be reapplied can take very long time (minutes or even hours). Similar problem can happen with unmount taking hours when there are hundreds of thousands of dirty inode in memory that need to be flushed to disk.

(http://oss.sgi.com/pipermail/xfs/2015-October/044457.html)

Q: Which I/O scheduler for XFS?

On rotational disks without hardware raid

CFQ: not great for XFS parallelism:

```
< dchinner> it doesn't allow other threads to get IO issued immediately after the first one
< dchinner> it waits, instead, for a timeslice to expire before moving to the IO of a different proc
< dchinner> so instead of interleaving the IO of multiple jobs in a single sweep across the disk,
            it enforces single threaded access to the disk
```

deadline: good option, doesn't have such problem

Note that some kernels have block multiqueue enabled which (currently - 08/2016) doesn't support I/O schedulers at all thus there is no optimisation and reordering IO for best seek order, so disable blk-mq for rotational disks (see CONFIG_SCSI_MQ_DEFAULT, CONFIG_DM_MQ_DEFAULT options and use_blk_mq parameter for scsi-mod/dm-mod kernel modules).

Also hardware raid can be smart enough to cache and reorder I/O requests thus additional layer of reordering (like Linux I/O scheduler) can potentially conflict and make performance worse. If you have such raid card then try method described below.

SSD disks or rotational disks but with hardware raid card that has cache enabled

Block multiqueue enabled (and thus no I/O scheduler at all) or block multiqueue disabled and *noop* or *deadline* I/O scheduler activated is good solution. SSD disks don't really need I/O schedulers while smart raid cards do I/O ordering on their own.

Note that if your raid is very dumb and/or has no cache enabled then it likely cannot reorder I/O requests and thus it could benefit from I/O scheduler.

Q: Why does userspace say "filesystem uses v1 dirs, limited functionality provided?"

Either you have a very old or a rather new filesystem coupled with too-old userspace.

Very old filesystems used a format called "directory version 1" and in this case the error is correct and hopefully self explanatory.

However, if you have a newer filesystem with version 5 superblocks and the metadata CRC feature enabled, older releases of xfsprogs may incorrectly issue the "v1 dir" message. In this case, get newer xfsprogs; at least v3.2.0, but preferably the latest release.

Retrieved from "http://xfs.org/index.php?title=XFS_FAQ&oldid=3004"

- This page was last modified on 5 December 2016, at 21:04.