Ext4 Filesystem

Ext4 is an an advanced level of the ext3 filesystem which incorporates scalability and reliability enhancements for supporting large filesystems (64 bit) in keeping with increasing disk capacities and state-of-the-art feature requirements.

Mailing list: linux-ext4@vger.kernel.org
Web site: http://ext4.wiki.kernel.org

1. Quick usage instructions:

Note: More extensive information for getting started with ext4 can be found at the ext4 wiki site at the URL:
http://ext4.wiki.kernel.org/index.php/Ext4_Howto

- Compile and install the latest version of e2fsprogs (as of this writing version 1.41.3) from:

  http://sourceforge.net/project/showfiles.php?group_id=2406

  or

  ftp://ftp.kernel.org/pub/linux/kernel/people/tytso/e2fsprogs/

  or grab the latest git repository from:

  git://git.kernel.org/pub/scm/fs/ext2/e2fsprogs.git

- Note that it is highly important to install the mke2fs.conf file that comes with the e2fsprogs 1.41.x sources in /etc/mke2fs.conf. If you have edited the /etc/mke2fs.conf file installed on your system, you will need to merge your changes with the version from e2fsprogs 1.41.x.

- Create a new filesystem using the ext4 filesystem type:

  ```
  # mke2fs -t ext4 /dev/hda1
  ```

  Or to configure an existing ext3 filesystem to support extents:

  ```
  # tune2fs -O extents /dev/hda1
  ```

  If the filesystem was created with 128 byte inodes, it can be converted to use 256 byte for greater efficiency via:

  ```
  # tune2fs -I 256 /dev/hda1
  ```

  (Note: we currently do not have tools to convert an ext4 filesystem back to ext3; so please do not do try this on production filesystems.)

- Mounting:

```
# mount -t ext4 /dev/hda1 /wherever
```

- When comparing performance with other filesystems, it's always important to try multiple workloads; very often a subtle change in a workload parameter can completely change the ranking of which filesystems do well compared to others. When comparing versus ext3, note that ext4 enables write barriers by default, while ext3 does not enable write barriers by default. So it is useful to use explicitly specify whether barriers are enabled or not when via the '-o barriers=[0|1]' mount option for both ext3 and ext4 filesystems for a fair comparison. When tuning ext3 for best benchmark numbers, it is often worthwhile to try changing the data journaling mode; '-o data=writeback,nobh' can be faster for some workloads. (Note however that running mounted with data=writeback can potentially leave stale data exposed in recently written files in case of an unclean shutdown, which could be a security exposure in some situations.) Configuring the filesystem with a large journal can also be helpful for metadata-intensive workloads.

2. Features

2.1 Currently available

* ability to use filesystems > 16TB (e2fsprogs support not available yet)
* extent format reduces metadata overhead (RAM, IO for access, transactions)
* extent format more robust in face of on-disk corruption due to magics,
* internal redundancy in tree
* improved file allocation (multi-block alloc)
* lift 32000 subdirectory limit imposed by i_links_count[1]
* nsec timestamps for mtime, atime, ctime, create time
* inode version field on disk (NFSv4, Lustre)
* reduced e2fsck time via uninit_bg feature
* journal checksumming for robustness, performance
* persistent file preallocation (e.g for streaming media, databases)
* ability to pack bitmaps and inode tables into larger virtual groups via the flex_bg feature
* large file support
* Inode allocation using large virtual block groups via flex_bg
* delayed allocation
* large block (up to pagesize) support
* efficent new ordered mode in JBD2 and ext4(avoid using buffer head to force the ordering)

[1] Filesystems with a block size of 1k may see a limit imposed by the directory hash tree having a maximum depth of two.

2.2 Candidate features for future inclusion

* Online defrag (patches available but not well tested)
* reduced mke2fs time via lazy itable initialization in conjuction with the uninit_bg feature (capability to do this is available in e2fsprogs but a kernel thread to do lazy zeroing of unused inode table blocks after filesystem is first mounted is required for safety)

There are several others under discussion, whether they all make it in is partly a function of how much time everyone has to work on them. Features like metadata checksumming have been discussed and planned for a bit but no patches exist yet so I'm not sure they're in the near-term roadmap.

The big performance win will come with mballoc, delalloc and flex_bg grouping of bitmaps and inode tables. Some test results available here:

- http://www.bullopensource.org/ext4/20080818-ffsb/ffsb-write-2.6.27-rc1.html
- http://www.bullopensource.org/ext4/20080818-ffsb/ffsb-readwrite-2.6.27-rc1.html

3. Options

When mounting an ext4 filesystem, the following option are accepted:
(*) == default

ro		Mount filesystem read only. Note that ext4 will replay the journal (and thus write to the partition) even when mounted "read only". The mount options "ro,noload" can be used to prevent writes to the filesystem.
journal_checksum		Enable checksumming of the journal transactions. This will allow the recovery code in e2fsck and the kernel to detect corruption in the kernel. It is a compatible change and will be ignored by older kernels.
journal_async_commit		Commit block can be written to disk without waiting for descriptor blocks. If enabled older kernels cannot mount the device. This will enable 'journal_checksum' internally.
journal=update		Update the ext4 file system's journal to the current format.
journal_dev=devnum		When the external journal device's major/minor numbers have changed, this option allows the user to specify the new journal location. The journal device is identified through its new major/minor numbers encoded in devnum.
norecovery noload		Don't load the journal on mounting. Note that if the filesystem was not unmounted cleanly, skipping the journal replay will lead to the filesystem containing inconsistencies that can lead to any number of problems.
data=journal		All data are committed into the journal prior to being written into the main file system.
data=ordered	(*)	All data are forced directly out to the main file system prior to its metadata being committed to the journal.

data=writeback		Data ordering is not preserved, data may be written into the main file system after its metadata has been committed to the journal.
commit=nrsec	(*)	Ext4 can be told to sync all its data and metadata every 'nrsec' seconds. The default value is 5 seconds. This means that if you lose your power, you will lose as much as the latest 5 seconds of work (your filesystem will not be damaged though, thanks to the journaling). This default value (or any low value) will hurt performance, but it's good for data-safety. Setting it to 0 will have the same effect as leaving it at the default (5 seconds). Setting it to very large values will improve performance.
barrier=<0|1(*)> barrier(*) nobarrier		This enables/disables the use of write barriers in the jbd code. barrier=0 disables, barrier=1 enables. This also requires an IO stack which can support barriers, and if jbd gets an error on a barrier write, it will disable again with a warning. Write barriers enforce proper on-disk ordering of journal commits, making volatile disk write caches safe to use, at some performance penalty. If your disks are battery-backed in one way or another, disabling barriers may safely improve performance. The mount options "barrier" and "nobarrier" can also be used to enable or disable barriers, for consistency with other ext4 mount options.
inode_readahead_blks=n		This tuning parameter controls the maximum number of inode table blocks that ext4's inode table readahead algorithm will pre-read into the buffer cache. The default value is 32 blocks.
orlov	(*)	This enables the new Orlov block allocator. It is enabled by default.
oldalloc		This disables the Orlov block allocator and enables the old block allocator. Orlov should have better performance - we'd like to get some feedback if it's the contrary for you.
user_xattr		Enables Extended User Attributes. Additionally, you need to have extended attribute support enabled in the kernel configuration (CONFIG_EXT4_FS_XATTR). See the attr(5) manual page and http://acl.bestbits.at/ to learn more about extended attributes.
nouser_xattr		Disables Extended User Attributes.
acl		Enables POSIX Access Control Lists support. Additionally, you need to have ACL support enabled in the kernel configuration (CONFIG_EXT4_FS_POSIX_ACL). See the acl(5) manual page and http://acl.bestbits.at/

for more information.

noacl		This option disables POSIX Access Control List support.
reservation		
noreservation		
bsddf	(*)	Make 'df' act like BSD.
minixdf		Make 'df' act like Minix.
debug		Extra debugging information is sent to syslog.
abort		Simulate the effects of calling ext4_abort() for debugging purposes. This is normally used while remounting a filesystem which is already mounted.
errors=remount-ro		Remount the filesystem read-only on an error.
errors=continue		Keep going on a filesystem error.
errors=panic		Panic and halt the machine if an error occurs. (These mount options override the errors behavior specified in the superblock, which can be configured using tune2fs)
data_err=ignore	(*)	Just print an error message if an error occurs in a file data buffer in ordered mode.
data_err=abort		Abort the journal if an error occurs in a file data buffer in ordered mode.
grpid		Give objects the same group ID as their creator.
bsdgroups		
nogrpid	(*)	New objects have the group ID of their creator.
sysvgroups		
resgid=n		The group ID which may use the reserved blocks.
resuid=n		The user ID which may use the reserved blocks.
sb=n		Use alternate superblock at this location.
quota		These options are ignored by the filesystem. They
noquota		are used only by quota tools to recognize volumes
grpquota		where quota should be turned on. See documentation
usrquota		in the quota-tools package for more details (http://sourceforge.net/projects/linuxquota).
jqfmt=<quota type>		These options tell filesystem details about quota
usrjquota=<file>		so that quota information can be properly updated
grpjquota=<file>		during journal replay. They replace the above quota options. See documentation in the quota-tools package for more details (http://sourceforge.net/projects/linuxquota).
bh	(*)	ext4 associates buffer heads to data pages to

nobh		(a) cache disk block mapping information (b) link pages into transaction to provide ordering guarantees. "bh" option forces use of buffer heads. "nobh" option tries to avoid associating buffer heads (supported only for "writeback" mode).
stripe=n		Number of filesystem blocks that mballoc will try to use for allocation size and alignment. For RAID5/6 systems this should be the number of data disks * RAID chunk size in file system blocks.
delalloc	(*)	Defer block allocation until just before ext4 writes out the block(s) in question. This allows ext4 to better allocation decisions more efficiently.
nodelalloc		Disable delayed allocation. Blocks are allocated when the data is copied from userspace to the page cache, either via the write(2) system call or when an mmap'ed page which was previously unallocated is written for the first time.
max_batch_time=usec		Maximum amount of time ext4 should wait for additional filesystem operations to be batch together with a synchronous write operation. Since a synchronous write operation is going to force a commit and then a wait for the I/O complete, it doesn't cost much, and can be a huge throughput win, we wait for a small amount of time to see if any other transactions can piggyback on the synchronous write. The algorithm used is designed to automatically tune for the speed of the disk, by measuring the amount of time (on average) that it takes to finish committing a transaction. Call this time the "commit time". If the time that the transaction has been running is less than the commit time, ext4 will try sleeping for the commit time to see if other operations will join the transaction. The commit time is capped by the max_batch_time, which defaults to 15000us (15ms). This optimization can be turned off entirely by setting max_batch_time to 0.
min_batch_time=usec		This parameter sets the commit time (as described above) to be at least min_batch_time. It defaults to zero microseconds. Increasing this parameter may improve the throughput of multi-threaded, synchronous workloads on very fast disks, at the cost of increasing latency.
journal_ioprio=prio		The I/O priority (from 0 to 7, where 0 is the highest priorty) which should be used for I/O operations submitted by kjournald2 during a commit operation. This defaults to 3, which is a slightly higher priority than the default I/O

priority.

auto_da_alloc(*) Many broken applications don't use fsync() when
noauto_da_alloc replacing existing files via patterns such as
fd = open("foo.new")/write(fd,..)/close(fd)/
rename("foo.new", "foo"), or worse yet,
fd = open("foo", O_TRUNC)/write(fd,..)/close(fd).
If auto_da_alloc is enabled, ext4 will detect
the replace-via-rename and replace-via-truncate
patterns and force that any delayed allocation
blocks are allocated such that at the next
journal commit, in the default data=ordered
mode, the data blocks of the new file are forced
to disk before the rename() operation is
committed. This provides roughly the same level
of guarantees as ext3, and avoids the
"zero-length" problem that can happen when a
system crashes before the delayed allocation
blocks are forced to disk.

discard Controls whether ext4 should issue discard/TRIM
nodiscard(*) commands to the underlying block device when
blocks are freed. This is useful for SSD devices
and sparse/thinly-provisioned LUNs, but it is off
by default until sufficient testing has been done.

Data Mode
=========
There are 3 different data modes:

* writeback mode
In data=writeback mode, ext4 does not journal data at all. This mode provides a similar level of journaling as that of XFS, JFS, and ReiserFS in its default mode - metadata journaling. A crash+recovery can cause incorrect data to appear in files which were written shortly before the crash. This mode will typically provide the best ext4 performance.

* ordered mode
In data=ordered mode, ext4 only officially journals metadata, but it logically groups metadata information related to data changes with the data blocks into a single unit called a transaction. When it's time to write the new metadata out to disk, the associated data blocks are written first. In general, this mode performs slightly slower than writeback but significantly faster than journal mode.

* journal mode
data=journal mode provides full data and metadata journaling. All new data is written to the journal first, and then to its final location.
In the event of a crash, the journal can be replayed, bringing both data and metadata into a consistent state. This mode is the slowest except when data needs to be read from and written to disk at the same time where it outperforms all others modes. Currently ext4 does not have delayed allocation support if this data journalling mode is selected.

References
==========

kernel source: <file:fs/ext4/>
<file:fs/jbd2/>

programs: http://e2fsprogs.sourceforge.net/

useful links: http://fedoraproject.org/wiki/ext3-devel
http://www.bullopensource.org/ext4/
http://ext4.wiki.kernel.org/index.php/Main_Page
http://fedoraproject.org/wiki/Features/Ext4