

Anatomy of ext4

Get to know the fourth extended file system

Skill Level: Intermediate

M. Tim Jones (mtj@mtjones.com)
Independent Author

17 Feb 2009

The fourth extended file system, or *ext4*, is the next generation of journaling file systems, retaining backward compatibility with the previous file system, *ext3*. Although *ext4* is not currently the standard, it will be the next default file system for most Linux® distributions. Get to know *ext4*, and discover why it will be your new favorite file system.

With every Linux kernel release come a few surprises, and this December's 2.6.28 release was no exception. This release is the first of a stable *ext4* file system (among a variety of other cool things, such as the *Btrfs*, which is still under heavy development). This next generation of the extended file system provides improved scalability, reliability, and considerable new functionality. *Ext4* is so scalable that the maximum file system would consume one million 1-terabyte (TB) disks.

A short history of the extended file system

The virtual file system switch

The VFS is a layer that abstracts the details of the underlying file systems from the upper-layer file system users. In doing this, the VFS allows Linux to support many file systems—simultaneously—on a given Linux system.

The first supported file system for Linux was the Minix file system. This file system had some significant performance issues, so another file system was created specifically for Linux called the *extended file system*. The first extended file system

(ext) was designed by Remy Card and introduced into Linux in April 1992. The ext file system was the first to use the virtual file system (VFS) switch implemented in the 0.96c kernel and supported file systems up to 2 gigabytes (GB) in size.

The second extended file system (ext2), also implemented by Remy Card, was introduced in January 1993. It adopted advanced ideas from other file systems of the day (such as the Berkeley Fast File System [FFS]). Ext2 extended supported file systems of 2TB in size, although 2.6 kernels extended the maximum size of ext2 file systems to 32TB.

Read more by Tim Jones on developerWorks

- [Tim's *Anatomy of...* articles](#)
- [All of Tim's articles on developerWorks](#)

The third extended file system (ext3) was a major advance in Linux file systems, even though its performance was less than some of its competitors. The ext3 file system introduced the concept of *journaling* to improve the reliability of the file system when the system is abruptly halted. And although competing file systems had better performance (such as Silicon Graphics' XFS and the IBM® Journaled File System [JFS]), ext3 supported in-place upgrades from systems already using ext2. Ext3 was introduced in November 2001 and implemented by Stephen Tweedie.

Fast-forward to today. We now have the fourth extended file system (ext4). Ext4 introduces numerous new advancements for performance, scalability, and reliability. Most notably, ext4 supports file systems of 1 exabyte in size. Ext4 was implemented by a team of developers, led by Theodore Tso (the ext3 maintainer), and was introduced in the 2.6.19 kernel. It is now stable in the 2.6.28 kernel (as of December 2008).

Ext4 borrows many useful concepts from a variety of competing file systems. For example, the extent approach to block management had been implemented in JFS. Another block management-related feature (delayed allocation) was implemented in both XFS and in Sun Microsystems' ZFS.

You'll find various improvements and innovations in the new ext4 file system. The improvements cover a number of feature angles from new functionality (new features), scalability (scaling beyond current file system constraints), reliability (in the face of failures), and of course, performance.

Functionality

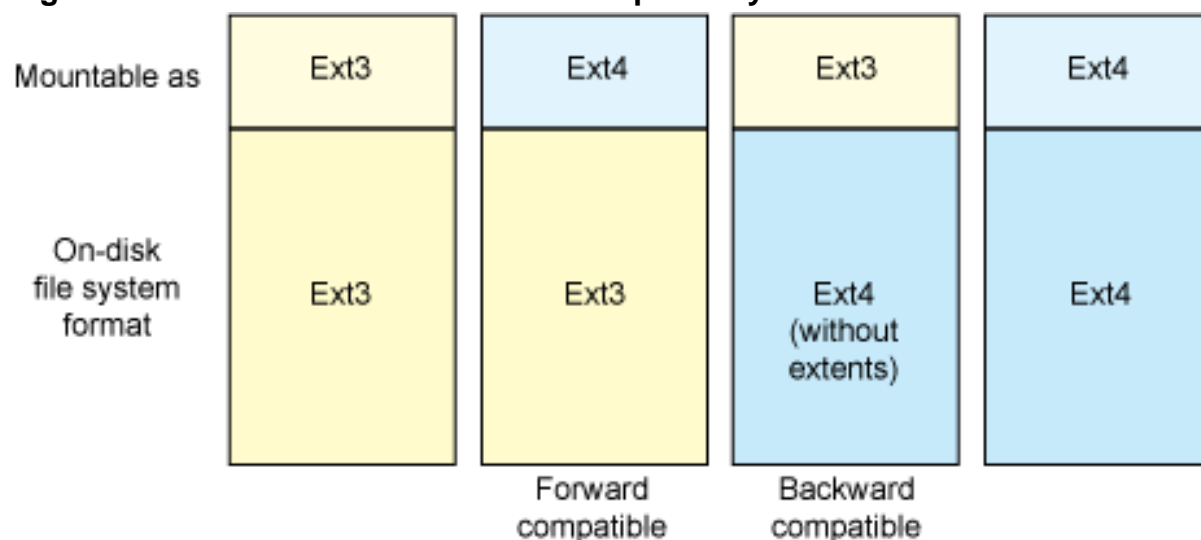
Ext4 includes a large amount of new functionality, but most important is its backward and forward compatibility with ext3 and improvements in timestamps that look to the

future of higher-performing Linux systems.

Forward and backward compatibility

As ext3 is one of the most popular file systems in use by Linux today, migrating to ext4 should be simple and painless. For this reason, ext4 was designed to be both forward and backward (to an extent) compatible (see Figure 1). Ext4 is forward compatible in that you can mount an ext3 file system as an ext4 file system. To fully take advantage of ext4, a file system migration is necessary to convert and exploit the new ext4 format. You can also mount an ext4 file system as ext3 (backward compatible), but only if the ext4 file system does not use extents (which is discussed in the performance section).

Figure 1. The forward and backward compatibility of ext4



In addition to the compatibility features, when you want to migrate an ext3 file system to ext4, you can do so gradually. This means that old files that you have not moved can remain in the older ext3 format, while new files (or older files that have been copied) will occupy the new ext4 data structures. In this way, you can migrate an ext3 file system online to an ext4 file system.

Improving timestamp resolution and range

Amazingly, timestamps in the extended file system arena prior to ext4 were seconds based. This was satisfactory in many settings, but as processors evolve with higher speeds and greater integration (multi-core processors) and Linux finds itself in other application domains such as high-performance computing, the seconds-based timestamp fails in its own simplicity. Ext4 has essentially future-proofed timestamps by extending them into a nanosecond LSB. The time range has also been extended with two additional bits to increase the lifetime by another 500 years.

Scalability

One of the most important aspects of file systems moving forward is their ability to scale given the growing demands placed on them. Ext4 accomplishes this in a number of ways, going beyond ext3 limits and breaking new ground in areas of file system metadata management.

Extending file system limits

One of the first visible differences in ext4 is the increased support for file system volumes, file sizes, and subdirectory limits. Ext4 supports file systems of up to 1 exabyte in size (1000 petabytes). Although that seems huge by today's standards, storage consumption continues to grow, so ext4 was definitely developed with the future in mind. Files within ext4 may be up to 16TB in size (assuming 4KB blocks), which is eight times the limit in ext3.

Finally, the subdirectory limit was extended with ext4, from 32KB directories deep to virtually unlimited. That may appear extreme, but one needs to consider the hierarchy of a file system that consumes an exabyte of storage. Directory indexing was also optimized to a hashed B-tree-like structure, so although the limits are much greater, ext4 supports very fast lookup times.

Extents

One of the primary disadvantages of ext3 was its method of allocation. Files were allocated using a bit map of free space, which was not very fast nor very scalable. Ext3's format is very efficient for small files but horribly inefficient for large files. Ext4 replaces ext3's mechanism with extents to improve allocation and support a more efficient storage structure. An *extent* is simply a way to represent a contiguous sequence of blocks. In doing this, metadata shrinks, because instead of maintaining information about where a block is stored, the extent maintains information about where a long list of contiguous blocks is stored (thus reducing the overall metadata storage).

Extents in ext4 adopt a layered approach to efficiently represent small files as well as extent trees to efficiently represent large files. For example, a single ext4 inode has sufficient space to reference four extents (where each extent represents a set of contiguous blocks). For large files (including those that are fragmented), an inode can reference an index node, each of which can reference a leaf node (referencing multiple extents). This constant depth extent tree provides a rich representation scheme for large, potentially sparse files. The nodes also include self-checking mechanisms to further protect against file system corruption.

Performance

One of the most important attributes used to measure new file systems is their fundamental performance. This is also one of the most difficult areas, as when a file system grows to massive sizes and expectations are placed on it to be highly reliable, performance can ultimately suffer. But even as ext4 addresses scalability and reliability, it provides a number of enhancements for improved performance.

File-level preallocation

Certain applications, such as databases or content streaming, rely on files to be stored in contiguous blocks (to exploit sequential block read optimization of drives as well as to maximize Read command-to-block ratios). Although extents can provide segments of contiguous blocks, another brute-force method is to preallocate very large sections of contiguous blocks in the size desired (as was implemented in the past with XFS). Ext4 implements this through a new system call that preallocates and initializes a file of a given size. You can then write the necessary data and provide bounded Read performance over the data.

Delaying block allocation

Another file size-based optimization is *delayed allocation*. This performance optimization delays the allocation of physical blocks on the disk until they are to be flushed to the disk. The key to this optimization is that by delaying the allocation of physical blocks until they need to be written to the disk, more blocks are present to allocate and write in contiguous blocks. This is similar to persistent preallocation except that the file system performs the task automatically. But if the size of the file is known beforehand, persistent preallocation is the best route.

Multi-block allocation

A final optimization—again, contiguous block related—is the *block allocator* for ext4. In ext3, the block allocator worked by allocating a single block at a time. When multiple blocks were necessary, it was possible to find contiguous data in non-contiguous blocks. Ext4 fixes this with a block allocator that allocates multiple blocks at a time, likely contiguous on disk. Like the previous optimizations, this optimization collects related data on the disk to optimize for sequential Read optimization.

The other aspect of multi-block allocation is the amount of processing required for allocating the blocks. Recall that ext3 performed allocation one block at a time. In the simplest units, that required a call to block allocation for each block. Allocating multiple blocks at a time requires many fewer calls to the block allocator, resulting in faster allocation and reduced processing.

Reliability

As file systems scale to the massive sizes possible with ext4, greater reliability concerns will certainly follow. Ext4 includes numerous self-protection and self-healing mechanisms to address this.

Checksumming the file system journal

Like ext3, ext4 is a journaling file system. *Journaling* is the process of logging changes to the file system through a *journal* (which is a dedicated circular log on a contiguous region of the disk). Actual changes to the physical storage are then performed from the log, which can more reliably implement the changes and ensure consistency, even if the system crashes or power is lost during the operation. The result is a reduced chance of file system corruption.

But even with journaling, corruption is still possible if erroneous entries find their way into the journal. To combat this, ext4 implements checksumming of the journal to ensure that valid changes make their way to the underlying file system. You can find additional references to journaling—an important aspect of ext4—in the [Resources](#) section.

Ext4 supports multiple modes of journaling, depending upon the needs of the user. For example, ext4 supports a mode in which only metadata is journaled (Writeback mode), a mode in which metadata is journaled but data is written as the metadata is written from the journal (Ordered mode), and a mode in which both metadata and data are journaled (Journal mode—the most reliable mode). Note that Journal mode, although the best way to ensure a consistent file system, is also the slowest, because all data flows through the journal.

Online defragmentation

Although ext4 incorporates features to reduce fragmentation within the file system (extents for sequential block allocation), some amount of fragmentation is impossible to avoid when a file system exists for a long period of time. For this reason, an online defragmentation tool exists to defragment both the file system and individual files for improved performance. The online defragmenter is a simple tool that copies files into a new ext4 inode that refers to contiguous extents.

The other aspect of online defragmentation is the reduced time required for a file system check (`fsck`). Ext4 marks unused groups of blocks within the inode table to allow the `fsck` process to skip them entirely to speed the check process. When the operating system decides to validate the file system because of internal corruption (which is inevitable as file systems increase in size and distribution), ext4's overall design means improved overall reliability.

What's next?

The extended file system obviously has a long and rich history within Linux—from the first introduction to ext in 1992 to ext4 in 2008. It was the first file system designed specifically for Linux and has proved one of the most efficient, stable, and powerful file systems available. Ext4 has evolved with advancements in file system research, incorporating ideas from other novel file systems (such as the XFS, JFS, Reiser, and the IRON fault-tolerant file system techniques). Although it's too early to know what's coming for ext5, it's clear that it will lead the way for enterprise-ready Linux systems.

Resources

Learn

- The article "[Ubuntu 9.04 Receives EXT4 Support](#)" finds very impressive performance improvements with ext4 in the latest Ubuntu release (compared with JFS, XFS, ReiserFS, and Ext3).
- Check out the [ext4 kernel wiki](#) (which will help you get ext4 running on your system), the [Fedora ext4 page](#), the [Kernel Newbies ext4 page](#), and of course, [Wikipedia](#) for more information on ext4. You can also read about all four of the extended file systems (1 through 4) on Wikipedia's [Extended file system](#) page, which provides links to file system comparisons and the history of the second extended file system.
- View a great presentation on ext4 titled "[Ext4: The Next Generation of Ext2/3 Filesystem](#)" from the IBM Linux Technology Center.
- Read Tim's "[Anatomy of the Linux file system](#)" (developerWorks, October 2007) and "[Anatomy of Linux journaling file systems](#)" (developerWorks, June 2008) for more information on Linux file systems and journaling file systems.
- Learn about the new kernel releases for 2.6.28 and 2.6.29 from Softpedia: [2.6.28](#) and Heise online: [2.6.29](#). The new kernel releases for 2.6.28 and 2.6.29 are important advances.
- Learn more about the [early history of ext4](#). This kernel trap article from 2006 includes the Theodore Tso's early proposal for what would become the ext4 file system.
- Read [Vijayan Prabhakaran's IRON File Systems dissertation](#) from the University of Wisconsin-Madison. The IRON (Internal RObustNess) techniques assume that disk drives fail in interesting ways and propose methods for managing them. In particular, IRON proposes failure detection and recovery techniques using journaling methods.
- Read more of [Tim's articles on developerWorks](#).
- In the [developerWorks Linux zone](#), find more resources for Linux developers (including developers who are [new to Linux](#)), and scan our [most popular articles and tutorials](#).
- See all [Linux tips](#) and [Linux tutorials](#) on developerWorks.
- Stay current with [developerWorks technical events and Webcasts](#).

Get products and technologies

- Download the latest kernel release from [kernel.org](#).
- With [IBM trial software](#), available for download directly from developerWorks,

build your next development project on Linux.

Discuss

- Get involved in the [developerWorks community](#) through blogs, forums, podcasts, and spaces.

About the author

M. Tim Jones

M. Tim Jones is an embedded firmware architect and the author of *Artificial Intelligence: A Systems Approach*, *GNU/Linux Application Programming* (now in its second edition), *AI Application Programming* (in its second edition), and *BSD Sockets Programming from a Multilanguage Perspective*. His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim is a Senior Architect for Emulex Corp. in Longmont, Colorado.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.