

AVANTIKA MATHUR, MINGMING CAO,  
AND ANDREAS DILGER

## ext4: the next generation of the ext3 file system



Avantika Mathur is a Linux kernel developer in the IBM Linux Technology Center. Her primary focus is ext3 and ext4 filesystem development and testing.

*mathur@us.ibm.com*



Mingming Cao has been a Linux kernel developer at the IBM Linux Technology Center for 7 years, mainly in the areas of IPC, block IO, and the ext3 filesystem. Her recent focus has been on bringing up the ext4 filesystem.

*cmm@us.ibm.com*



Andreas Dilger is a Principal Systems Software Engineer for Cluster Filesystems, Inc., designing and developing the Lustre distributed file system on some of the largest computers in the world. He started programming more than 25 years ago and has spent much of the last 10 years focused on Linux filesystem development.

*adilger@clusterfs.com*

LAST YEAR, A NEW LINUX FILE SYSTEM was born: ext4. A descendant of the ext3 file system, ext4 will soon replace ext3 as the “Linux file system.” Ext4 provides greater scalability and higher performance than ext3, while maintaining reliability and stability, giving users many reasons to switch to this new file system. The primary goal for ext4 is to support larger files and file systems. Once it is mature, the developing ext4 file system will be suitable for a greater variety of workloads, from desktop to enterprise solutions.

### Why Ext4

Among the many file systems Linux offers today, ext3 is the most popular, with the largest user base and development community. Having been designed with stability and maintenance in mind makes ext3 a very reliable file system with relatively good performance. Thus it has been the default file system in many commercial Linux distributions for many years.

However, the conservative design of ext3 limits its scalability and performance. One of the hard limits faced by ext3 today is the 16-TB filesystem size maximum. This limit has already been reached in large installations and will soon be hit by desktop users. Today, 1-TB external hard drives are readily available in stores, and disk capacity can double every year.

Last year, a series of patches were sent to the Linux kernel mailing list, to address filesystem capacity and to add extents mapping to ext3. The extent patches would cause intrusive changes to the on-disk format and break forward compatibility for file systems that used them. In order to maintain the stable ext3 file system for its massive user base, it was decided to fork the ext4 file system from ext3 and address performance and scalability issues in this new file system.

Why not use a file system such as XFS for this? The answer is that the XFS code in Linux is very complex, being burdened with an extra layer of compatibility code for IRIX. Even with the addition of the features described here, ext4 is still much smaller and more understandable than XFS (25k lines vs. 106k lines). Also, there is a considerable investment in the stability and robustness of the ext3 and e2fsck code base, most of which will continue to be used for ext4.

---

## What's New in Ext4

---

Ext4 was included in mainline Linux version 2.6.19. The file system is currently in development mode, titled `ext4dev`, explicitly warning users that it is not ready for production use. There are many new features in the `ext4` road map under development and testing. Some of the features in progress may continue to change the filesystem layout. Any future changes, as with the current ones, will be protected by appropriate feature flags so that existing kernels and `e2fsprogs` will be able to know whether it is safe to mount a given `ext4` file system. Once the layout is finalized, `ext4` will be converted from development to stable mode. At that point, `ext4` will be stable and available for general use by all users in need of a more scalable and modern version of `ext3`.

The initial version of the `ext4` file system includes two new key features: extent support and 48-bit block numbers. Combined, these features support larger file systems and better performance on large files.

---

### EXTENT SUPPORT

---

The `ext3` file system uses the traditional indirect block mapping scheme, which is efficient for small or sparse files but causes high metadata overhead and poor performance when dealing with large files, especially on delete and truncate operations. To address this issue, `ext4` uses extent mapping as an efficient way to represent large contiguous files.

An extent is a single descriptor that represents a range of contiguous blocks. A single extent in `ext4` can represent up to 128 MB. Four extents can be stored directly in the inode structure. That is generally sufficient for small to medium contiguous files, but for very large or highly fragmented files, an extents tree is created to efficiently look up the many extents required.

Extents mapping improves performance on large files, such as mp3, DVD, video, or database files, as it is tuned toward allocating large contiguous blocks. Extents bring about a 25% throughput gain in large sequential I/O workloads when compared with `ext3`. A similar performance gain was also seen on the Postmark benchmark, which simulates a mail server, with a large number of small to medium files. With extents the metadata is more compact, causing greatly reduced CPU usage.

Although the indirect block and extents mapping schemes are incompatible, both are supported by `ext4`, and files can be converted between the two formats. This is discussed further in the migration section.

---

### LARGE FILE SYSTEM SUPPORT

---

The first issue addressed in `ext4` is the filesystem capacity limit. Because `ext3` uses 32 bits to represent block numbers and has a default 4k block size, the file system is limited to a maximum of 16 TB. `Ext4` uses 48-bit block numbers. In theory this allows it to support 1-EB (1-million-TB) file systems. This change was made in combination with the extents patches, which use 48-bit physical block numbers in the extents structure. Other metadata changes, such as in the super-block structure, were also made to support the 48-bit block number.

In order to support more than 32-bit block numbers in the journaling block layer (JBD), JBD2 was forked from JBD at the same time that `ext4`

was cloned. There is also work underway to add checksumming to the journal in JBD2 to validate this critical metadata at recovery time. Currently, JBD2 is only used by ext4, but eventually both 32-bit and 64-bit Linux file systems will be able to use JBD2 for journaling support.

One may question why we chose 48-bit block numbers rather than the round 64 bits. Although it is possible to design for 64-bit ext4 file systems, this is impractical today because of reliability and serviceability problems. Having full 64-bit physical and logical block numbers would have meant that fewer extents could fit within the inode (2 vs. 4) for only very theoretical gains. It would take 119 years at today's speeds to run `e2fsck` on even a  $2^{48}$ -block file system. The ext4 developers will focus their efforts on improving the reliability aspects before worrying about a theoretical size limit.

### What's Next

The increased file system capacity created by the 48-bit block numbers and extent mapping features that are currently in ext4 will provide many new features in the road map. These features are focused on enhancing ext4 scalability, reliability, block placement, and performance.

An ext4 git tree is hosted at [git://git.kernel.org/pub/scm/linux/kernel/git/tytso/ext4](http://git.kernel.org/pub/scm/linux/kernel/git/tytso/ext4). The tree contains the series of patches in line for ext4. Up-to-date information on new ext4 features, patch sets, and development discussion can be found at the ext4 wiki page, <http://ext4.wiki.kernel.org/>.

### BLOCK ALLOCATION ENHANCEMENTS

Fragmentation is a key factor affecting filesystem performance. The following features in the ext4 road map attempt to address performance by avoiding or decreasing fragmentation. This is essential for the efficient use of extents and maximizing performance on modern high-bandwidth storage.

#### PERSISTENT PREALLOCATION

Applications such as large databases often write zeros to a file for guaranteed and contiguous filesystem space reservation. Persistent preallocation in ext4 allocates a contiguous set of blocks for a file without the expensive zero-out. The preallocated extents contain a flag specifying that the blocks are uninitialized. These uninitialized extents are protected from undesired exposure of their contents through read operations. A new system call, `sys_fallocate`, is planned to be added to the Linux kernel, and the existing `posix_fallocate` library is being modified. These interfaces can be used by users to specify the portion of the file to preallocate.

#### DELAYED ALLOCATION AND MULTIPLE BLOCK ALLOCATION

The ext3 block allocation scheme is not very efficient for allocating extents, as blocks are allocated one by one during write operations, so ext4 will use delayed allocation and multiple block mechanisms to efficiently allocate many blocks at a time and contribute to avoiding fragmentation. With delayed allocation, block allocations are deferred to page flush time, rather than during the write operation. This avoids unnecessary block allocation for short-lived files and provides the opportunity to queue many individual block allocation requests into a single request.

The multiple block allocation feature uses a buddy data structure to efficiently locate free extents and allocates an entire extent referencing multiple blocks, rather than allocating one at a time. Combined, delayed allocation and multiple block allocation have been shown to significantly reduce CPU usage and improve throughput on large I/O. Performance testing shows a 30% throughput gain for large sequential writes and 7–10% improvement on small files (e.g., those seen in mail-server-type workloads).

#### **ONLINE DEFRAGMENTATION**

Even though there are techniques in place to attempt to avoid file fragmentation, with age, a file system can still become highly fragmented. The online defragmentation tool is designed to defragment individual files or an entire file system. The defragmentation is performed by creating a temporary inode, using multiple block allocation to allocate contiguous blocks to the inode, reading all data from the original file to the page cache, then flushing the data to disk and migrating the newly allocated blocks over to the original inode.

#### **SCALABILITY ENHANCEMENTS**

Besides enlarging the overall file system capacity, there are many other scalability features planned for ext4.

Although ext3 has support for different inode sizes, the default inode structure size is 128 bytes, with little extra room to support new features. In ext4, the default inode size will be enlarged to 256 bytes. This will provide space for the new fields needed for the planned features, nanosecond time stamps, and inode versioning. The latter is a counter incremented on each file modification that will be used by NFSv4 (network file system) to keep track of file updates.

By using a larger inode size in ext4, the EA-in-inode feature can be enabled by default. This feature is also available in ext3, but because of the smaller default inode size it is not widely used. EA-in-inode stores extended attributes (EAs) directly in the inode body, making EA access much faster. The faster EA access can greatly improve performance, sometimes by 2–3 times, for those using SELinux, ACLs, or other EAs. Additional EAs, which don't fit in the inode body, are stored in a single filesystem block. This limits the maximum EA capacity per file to 4k. In ext4, there is a plan to remove this limit by storing large EAs in a file.

To address directory scalability, the directory indexing feature, available in ext3, will be turned on by default in ext4. Directory indexing uses a specialized Btree-like structure to store directory entries, rather than a linked list with linear access times. This significantly improves performance on certain applications with very large directories, such as Web caches and mail systems using the Maildir format. Performance improvements were often by factors of 50–100, in particular for directories with more than 10,000 files. An additional scalability improvement is eliminating the 32,000 subdirectory limit in ext4.

Support for larger files, extending beyond the 2-TB limit, are in the road map. There is interest in efficiently supporting large numbers of files, surpassing the 4-billion limit, which implies 64-bit inode numbers and dynamic inode tables. Because of the potential intrusive on-disk format changes involved, plans and design aspects are still on the drawing board.

### FASTER REPAIR AND RECOVERY

A file system is not very useful if it cannot be repaired or recovered in a reasonable amount of time. As the filesystem size grows, the e2fsck time becomes unbearable, taking from minutes to years depending on the file-system size. This issue is more prevalent for ext4, as it can support larger file systems. Although the ext4 journaling support tries to avoid the need for file system recovery as much as possible, in the event of a disk error e2fsck is still necessary.

The e2fsck tool scans the whole file system, regardless of whether only a small part of it is being used. With a little more information about the file system, e2fsck could skip those unused block groups or inode structures. The uninitialized block groups feature uses a flag in the block group to mark whether it is uninitialized. Once it is written to, watermarks are used to keep track of the last used inode structures. Any uninitialized block groups or inodes are not scanned by e2fsck, which greatly reduces the run time. This feature can speed up e2fsck dramatically, reducing run times to 1/2 to 1/10 of the original run time, depending on how the file system is used. The flags marking a block group uninitialized and the high watermark are checksummed. If there is ever corruption, e2fsck will fall back to the slower, but safer, full scan for that block group. As new features are added to ext4, the user tools, e2fsprogs, will be updated correspondingly.

### Migration from Ext3 to Ext4

Compatibility between ext4 and existing ext3 file systems has been maintained as much as possible. Even though ext4 has changed some parts of the on-disk format, it is possible to take advantage of many of the ext4 features, such as extents, delayed allocation, multiple block allocation, and faster e2fsck, without requiring a backup and restore.

There is an upgrade path from ext3 that allows existing file systems to start using ext4 immediately, without requiring a lengthy downtime. Users can mount an existing ext3 file system as ext4, and all existing files are still treated as ext3 files. Any new files created in this ext4 file system will be extent-mapping-based. There is a flag in each individual file to indicate whether it is an ext3 file (indirect mapped) or ext4 file (extent mapped).

There is also a way to do a systemwide migration. A set of tools is under development to migrate an entire file system from ext3 to ext4, which includes transferring indirect files to extent files and enlarging the inode structure to 256 bytes. The first part can be performed online in combination with online defragmentation. This tool will perform the conversion from ext3 to ext4 while simultaneously defragmenting the file system and files, and it will have the option of converting only part of the file system. Resizing the inodes must be performed offline, and this can be done in conjunction with converting files to extent mapping. In this case the whole file system is scanned, and proper backup is done to prevent data loss if the system crashes during the migration.

Users who are hesitant to migrate to ext4 immediately can optionally format their ext3 file system with large inodes (256 bytes or more) to take advantage of the EA-in-inode feature today and nanosecond timestamps if they migrate to ext4. This would avoid the need to do an offline migration step to resize the inodes.

There is also a downgrade path from ext4 to ext3, with a method to convert the extent files back to indirect mapping files. In the case that users prefer to go back to ext3, they can mount the ext4 file system with the “noextents” mount option, copy the extent-based ext4 files to new files, rename these over the old extents, use tunefs to clear the INCOMPAT\_EXTENTS flag, and then remount as an ext3 file system.

---

## Conclusion

---

As we have discussed, a tremendous amount of work has gone into the ext4 file system, and this work is ongoing. As we stabilize the file system, ext4 will become suitable for production use, making it a good choice for a wide variety of workloads. What was once essentially a simple file system has turned into an enterprise-ready modern file system, with a good balance of scalability, reliability, performance, and stability. Eventually ext4 will replace ext3 as the default Linux file system.

---

## LEGAL STATEMENT

---

Copyright © 2007 IBM.

This work represents the view of the authors and does not necessarily represent the view of IBM.

IBM and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.

Lustre is a trademark of Cluster File Systems, Inc.

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

Other company, product, and service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

This document is provided “AS IS,” with no express or implied warranties. Use the information in this document at your own risk.