opensource.com

An introduction to Linux's EXT4 filesystem

By David Both (Correspondent)

22-28 minutes

In previous articles about Linux filesystems, I wrote an introduction to Linux filesystems and about some higher-level concepts such as everything is a file. I want to go into more detail about the specifics of the EXT filesystems, but first, let's answer the question, "What is a filesystem?" A filesystem is all of the following:

1. **Data storage:** The primary function of any filesystem is to be a structured place to store and retrieve data.
2. **Namespace:** A naming and organizational methodology that provides rules for naming and structuring data.
3. **Security model:** A scheme for defining access rights.
4. **API:** System function calls to manipulate filesystem

objects like directories and files.

5. **Implementation:** The software to implement the above.

   This article concentrates on the first item in the list and explores the metadata structures that provide the logical framework for data storage in an EXT filesystem.

EXT filesystem history

Although written for Linux, the EXT filesystem has its roots in the Minix operating system and the Minix filesystem, which predate Linux by about five years, being first released in 1987. Understanding the EXT4 filesystem is much easier if we look at the history and technical evolution of the EXT filesystem family from its Minix roots.

Minix

When writing the original Linux kernel, Linus Torvalds needed a filesystem but didn't want to write one then. So he simply included the Minix filesystem, which had been written by Andrew S. Tanenbaum and was a part of Tanenbaum's Minix operating system. Minix was a

Unix-like operating system written for educational purposes. Its code was freely available and appropriately licensed to allow Torvalds to include it in his first version of Linux.

Minix has the following structures, most of which are located in the partition where the filesystem is generated:

- A **boot sector** in the first sector of the hard drive on which it is installed. The boot block includes a very small boot record and a partition table.
- The first block in each partition is a **superblock** that contains the metadata that defines the other filesystem structures and locates them on the physical disk assigned to the partition.
- An **inode bitmap block**, which determines which inodes are used and which are free.
- The **inodes**, which have their own space on the disk. Each inode contains information about one file, including the locations of the data blocks, i.e., zones belonging to the file.
- A **zone bitmap** to keep track of the used and free data zones.
- A **data zone**, in which the data is actually stored.

For both types of bitmaps, one bit represents one specific data zone or one specific inode. If the bit is zero, the zone or inode is free and available for use, but if the bit is one, the data zone or inode is in use.

What is an inode? Short for index-node, an inode is a 256-byte block on the disk and stores data about the file. This includes the file's size; the user IDs of the file's user and group owners; the file mode (i.e., the access permissions); and three timestamps specifying the time and date that: the file was last accessed, last modified, and the data in the inode was last modified.

The inode also contains data that points to the location of the file's data on the hard drive. In Minix and the EXT1-3 filesystems, this is a list of data zones or blocks. The Minix filesystem inodes supported nine data blocks, seven direct and two indirect. If you'd like to learn more, there is an excellent PDF with a detailed description of the Minix filesystem structure and a quick overview of the inode pointer structure on Wikipedia.

EXT

The original EXT filesystem (Extended) was written by Rémy Card and released with Linux in 1992 to overcome some size limitations of the Minix filesystem.

The primary structural changes were to the metadata of the filesystem, which was based on the Unix filesystem (UFS), which is also known as the Berkeley Fast File System (FFS). I found very little published information about the EXT filesystem that can be verified, apparently because it had significant problems and was quickly superseded by the EXT2 filesystem.

EXT2

The EXT2 filesystem was quite successful. It was used in Linux distributions for many years, and it was the first filesystem I encountered when I started using Red Hat Linux 5.0 back in about 1997. The EXT2 filesystem has essentially the same metadata structures as the EXT filesystem, however EXT2 is more forward-looking, in that a lot of disk space is left between the metadata structures for future use.

Like Minix, EXT2 has a boot sector in the first sector of the hard drive on which it is installed, which includes a very small boot record and a partition table. Then there is some reserved space after the boot sector, which spans the space between the boot record and the first partition on the hard drive that is usually on the next cylinder boundary. GRUB2—and possibly GRUB1—

uses this space for part of its boot code.

The space in each EXT2 partition is divided into cylinder groups that allow for more granular management of the data space. In my experience, the group size usually amounts to about 8MB. Figure 1, below, shows the basic structure of a cylinder group. The data allocation unit in a cylinder is the block, which is usually 4K in size.

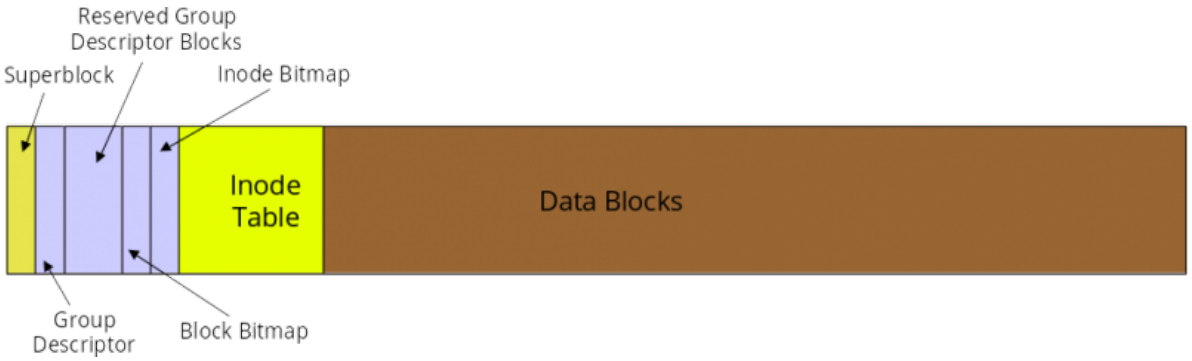

Figure 1: The structure of a cylinder group in the EXT filesystems

The first block in the cylinder group is a superblock, which contains the metadata that defines the other filesystem structures and locates them on the physical disk. Some of the additional groups in the partition will have backup superblocks, but not all. A damaged superblock can be replaced by using a disk utility such as **dd** to copy the contents of a backup superblock to the primary superblock. It does not happen often, but once, many years ago, I had a damaged superblock, and I was able to restore its contents using one of the

backup superblocks. Fortunately, I had been foresighted and used the **dumpe2fs** command to dump the descriptor information of the partitions on my system.

Following is the partial output from the **dumpe2fs** command. It shows the metadata contained in the superblock, as well as data about each of the first two cylinder groups in the filesystem.

```
# dumpe2fs /dev/sda1

Filesystem volume name:   boot

Last mounted on:          /boot

Filesystem UUID:          79fc5ed8-5bbc-4dfe-8359-
b7b36be6eed3

Filesystem magic number:  0xEF53

Filesystem revision #:    1 (dynamic)

Filesystem features:      has_journal ext_attr
resize_inode dir_index filetype needs_recovery extent
64bit flex_bg sparse_super large_file huge_file dir nlink
```

```
extra_isize

Filesystem flags:         signed_directory_hash

Default mount options:    user_xattr acl

Filesystem state:         clean

Errors behavior:          Continue

Filesystem OS type:       Linux

Inode count:              122160

Block count:              488192

Reserved block count:     24409

Free blocks:              376512

Free inodes:              121690

First block:              0

Block size:               4096
```

```
Fragment size:             4096
Group descriptor size:     64
Reserved GDT blocks:       238
Blocks per group:          32768
Fragments per group:       32768
Inodes per group:          8144
Inode blocks per group:    509
Flex block group size:     16
Filesystem created:        Tue Feb  7 09:33:34 2017
Last mount time:           Sat Apr 29 21:42:01 2017
Last write time:           Sat Apr 29 21:42:01 2017
Mount count:               25
```

```
Maximum mount count:      -1

Last checked:             Tue Feb  7 09:33:34 2017

Check interval:           0 (<none>)

Lifetime writes:          594 MB

Reserved blocks uid:      0 (user root)

Reserved blocks gid:      0 (group root)

First inode:              11

Inode size:               256

Required extra isize:     32

Desired extra isize:      32

Journal inode:            8

Default directory hash:   half_md4

Directory Hash Seed:      c780bac9-d4bf-4f35-
```

```
b695-0fe35e8d2d60

Journal backup:           inode blocks

Journal features:         journal_64bit

Journal size:             32M

Journal length:           8192

Journal sequence:         0x00000213

Journal start:            0


Group 0: (Blocks 0-32767)

 Primary superblock at 0, Group descriptors at 1-1

 Reserved GDT blocks at 2-239

 Block bitmap at 240 (+240)
```

Inode bitmap at 255 (+255)

Inode table at 270-778 (+270)

24839 free blocks, 7676 free inodes, 16 directories

Free blocks: 7929-32767

Free inodes: 440, 470-8144

Group 1: (Blocks 32768-65535)

Backup superblock at 32768, Group descriptors at 32769-32769

Reserved GDT blocks at 32770-33007

Block bitmap at 241 (bg #0 + 241)

Inode bitmap at 256 (bg #0 + 256)

Inode table at 779-1287 (bg #0 + 779)

8668 free blocks, 8142 free inodes, 2 directories

```
 Free blocks: 33008-33283, 33332-33791,
33974-33975, 34023-34092, 34094-34104,
34526-34687, 34706-34723, 34817-35374,
35421-35844, 35935-36355, 36357-36863,
38912-39935, 39940-40570, 42620-42623, 42655,
42674-42687, 42721-42751, 42798-42815, 42847,
42875-42879, 42918-42943, 42975, 43000-43007,
43519, 43559-44031, 44042-44543, 44545-45055,
45116-45567, 45601-45631, 45658-45663,
45689-45695, 45736-45759, 45802-45823,
45857-45887, 45919, 45950-45951, 45972-45983,
46014-46015, 46057-46079, 46112-46591,
46921-47103, 49152-49395, 50027-50355,
52237-52255, 52285-52287, 52323-52351, 52383,
52450-52479, 52518-52543, 52584-52607,
52652-52671, 52734-52735, 52743-53247

 Free inodes: 8147-16288

Group 2: (Blocks 65536-98303)

 Block bitmap at 242 (bg #0 + 242)

 Inode bitmap at 257 (bg #0 + 257)
```

Inode table at 1288-1796 (bg #0 + 1288)

6326 free blocks, 8144 free inodes, 0 directories

Free blocks: 67042-67583, 72201-72994, 80185-80349, 81191-81919, 90112-94207

Free inodes: 16289-24432

Group 3: (Blocks 98304-131071)

<snip>

Each cylinder group has its own inode bitmap that is used to determine which inodes are used and which are free within that group. The inodes have their own space in each group. Each inode contains information about one file, including the locations of the data blocks belonging to the file. The block bitmap keeps track of the used and free data blocks within the filesystem. Notice that there is a great deal of data about the filesystem in the output shown above. On very large filesystems the group data can run to hundreds of pages in length. The group metadata includes a listing

of all of the free data blocks in the group.

The EXT filesystem implemented data-allocation strategies that ensured minimal file fragmentation. Reducing fragmentation improved filesystem performance. Those strategies are described below, in the section on EXT4.

The biggest problem with the EXT2 filesystem, which I encountered on some occasions, was that it could take many hours to recover after a crash because the **fsck** (file system check) program took a very long time to locate and correct any inconsistencies in the filesystem. It once took over 28 hours on one of my computers to fully recover a disk upon reboot after a crash—and that was when disks were measured in the low hundreds of megabytes in size.

EXT3

The EXT3 filesystem had the singular objective of overcoming the massive amounts of time that the **fsck** program required to fully recover a disk structure damaged by an improper shutdown that occurred during a file-update operation. The only addition to the EXT filesystem was the journal, which records in advance the changes that will be performed to the

filesystem. The rest of the disk structure is the same as it was in EXT2.

Instead of writing data to the disk's data areas directly, as in previous versions, the journal in EXT3 writes file data, along with its metadata, to a specified area on the disk. Once the data is safely on the hard drive, it can be merged in or appended to the target file with almost zero chance of losing data. As this data is committed to the data area of the disk, the journal is updated so that the filesystem will remain in a consistent state in the event of a system failure before all the data in the journal is committed. On the next boot, the filesystem will be checked for inconsistencies, and data remaining in the journal will then be committed to the data areas of the disk to complete the updates to the target file.

Journaling does reduce data-write performance, however there are three options available for the journal that allow the user to choose between performance and data integrity and safety. My personal preference is on the side of safety because my environments do not require heavy disk-write activity.

The journaling function reduces the time required to check the hard drive for inconsistencies after a failure from hours (or even days) to mere minutes, at the

most. I have had many issues over the years that have crashed my systems. The details could fill another article, but suffice it to say that most were self-inflicted, like kicking out a power plug. Fortunately, the EXT journaling filesystems have reduced that bootup recovery time to two or three minutes. In addition, I have never had a problem with lost data since I started using EXT3 with journaling.

The journaling feature of EXT3 can be turned off and it then functions as an EXT2 filesystem. The journal itself still exists, empty and unused. Simply remount the partition with the mount command using the type parameter to specify EXT2. You may be able to do this from the command line, depending upon which filesystem you are working with, but you can change the type specifier in the **/etc/fstab** file and then reboot. I strongly recommend against mounting an EXT3 filesystem as EXT2 because of the additional potential for lost data and extended recovery times.

An existing EXT2 filesystem can be upgraded to EXT3 with the addition of a journal using the following command.

```
tune2fs -j /dev/sda1
```

Where **/dev/sda1** is the drive and partition identifier. Be sure to change the file type specifier in **/etc/fstab** and remount the partition or reboot the system to have the change take effect.

EXT4

The EXT4 filesystem primarily improves performance, reliability, and capacity. To improve reliability, metadata and journal checksums were added. To meet various mission-critical requirements, the filesystem timestamps were improved with the addition of intervals down to nanoseconds. The addition of two high-order bits in the timestamp field defers the Year 2038 problem until 2446—for EXT4 filesystems, at least.

In EXT4, data allocation was changed from fixed blocks to extents. An extent is described by its starting and ending place on the hard drive. This makes it possible to describe very long, physically contiguous files in a single inode pointer entry, which can significantly reduce the number of pointers required to describe the location of all the data in larger files. Other allocation strategies have been implemented in EXT4 to further reduce fragmentation.

EXT4 reduces fragmentation by scattering newly

created files across the disk so that they are not bunched up in one location at the beginning of the disk, as many early PC filesystems did. The file-allocation algorithms attempt to spread the files as evenly as possible among the cylinder groups and, when fragmentation is necessary, to keep the discontinuous file extents as close as possible to others in the same file to minimize head seek and rotational latency as much as possible. Additional strategies are used to pre-allocate extra disk space when a new file is created or when an existing file is extended. This helps to ensure that extending the file will not automatically result in its becoming fragmented. New files are never allocated immediately after existing files, which also prevents fragmentation of the existing files.

Aside from the actual location of the data on the disk, EXT4 uses functional strategies, such as delayed allocation, to allow the filesystem to collect all the data being written to the disk before allocating space to it. This can improve the likelihood that the data space will be contiguous.

Older EXT filesystems, such as EXT2 and EXT3, can be mounted as EXT4 to make some minor performance gains. Unfortunately, this requires turning off some of

the important new features of EXT4, so I recommend against this.

EXT4 has been the default filesystem for Fedora since Fedora 14. An EXT3 filesystem can be upgraded to EXT4 using the procedure described in the Fedora documentation, however its performance will still suffer due to residual EXT3 metadata structures. The best method for upgrading to EXT4 from EXT3 is to back up all the data on the target filesystem partition, use the **mkfs** command to write an empty EXT4 filesystem to the partition, and then restore all the data from the backup.

Inode

The inode, described previously, is a key component of the metadata in EXT filesystems. Figure 2 shows the relationship between the inode and the data stored on the hard drive. This diagram is the directory and inode for a single file which, in this case, may be highly fragmented. The EXT filesystems work actively to reduce fragmentation, so it is very unlikely you will ever see a file with this many indirect data blocks or extents. In fact, as you will see below, fragmentation is extremely low in EXT filesystems, so most inodes will

use only one or two direct data pointers and none of the indirect pointers.

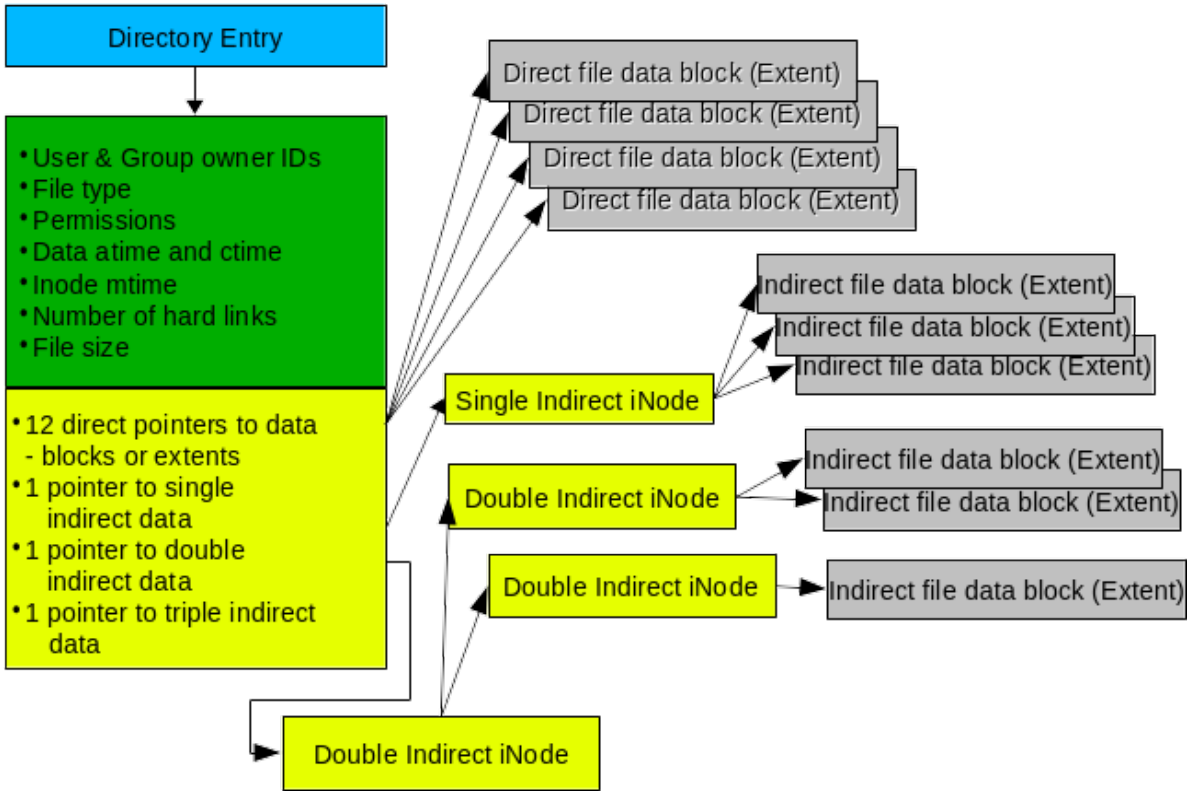

Figure 2: The inode stores information about each file and enables the EXT filesystem to locate all data belonging to it.

The inode does not contain the name of the file. Access to a file is via the directory entry, which itself is the name of the file and contains a pointer to the inode. The value of that pointer is the inode number. Each inode in a filesystem has a unique ID number, but inodes in other filesystems on the same computer (and even the same hard drive) can have the same inode number. This has implications for links, and this discussion is beyond the scope of this article.

The inode contains the metadata about the file, including its type and permissions as well as its size. The inode also contains space for 15 pointers that describe the location and length of data blocks or extents in the data portion of the cylinder group. Twelve of the pointers provide direct access to the data extents and should be sufficient to handle most files. However, for files that have significant fragmentation, it becomes necessary to have some additional capabilities in the form of indirect nodes. Technically these are not really inodes, so I use the term "node" here for convenience.

An indirect node is a normal data block in the filesystem that is used only for describing data and not for storage of metadata, thus more than 15 entries can be supported. For example, a block size of 4K can support 512 4-byte indirect nodes, allowing **12 (direct) + 512 (indirect) = 524** extents for a single file. Double and triple indirect node support is also supported, but most of us are unlikely to encounter files requiring that many extents.

Data fragmentation

For many older PC filesystems, such as FAT (and all its variants) and NTFS, fragmentation has been a

significant problem resulting in degraded disk performance. Defragmentation became an industry in itself with different brands of defragmentation software that ranged from very effective to only marginally so.

Linux's extended filesystems use data-allocation strategies that help to minimize fragmentation of files on the hard drive and reduce the effects of fragmentation when it does occur. You can use the **fsck** command on EXT filesystems to check the total filesystem fragmentation. The following example checks the home directory of my main workstation, which was only 1.5% fragmented. Be sure to use the **-n** parameter, because it prevents **fsck** from taking any action on the scanned filesystem.

```
fsck -fn /dev/mapper/vg_01-home
```

I once performed some theoretical calculations to determine whether disk defragmentation might result in any noticeable performance improvement. While I did make some assumptions, the disk performance data I used were from a new 300GB, Western Digital hard drive with a 2.0ms track-to-track seek time. The number of files in this example was the actual number that existed in the filesystem on the day I did the calculation. I did assume that a fairly large amount of

the fragmented files (20%) would be touched each day.

Total files	**271,794**
% fragmentation	5.00%
Discontinuities	13,590
% fragmented files touched per day	20% (assume)
Number of additional seeks	2,718
Average seek time	10.90 ms
Total additional seek time per day	29.63 sec
	0.49 min
Track-to-track seek time	2.00 ms
Total additional seek time per day	5.44 sec
	0.091 min

Table 1: The theoretical effects of fragmentation on disk performance

I have done two calculations for the total additional seek time per day, one based on the track-to-track seek time, which is the more likely scenario for most files

due to the EXT file allocation strategies, and one for the average seek time, which I assumed would make a fair worst-case scenario.

As you can see from Table 1, the impact of fragmentation on a modern EXT filesystem with a hard drive of even modest performance would be minimal and negligible for the vast majority of applications. You can plug the numbers from your environment into your own similar spreadsheet to see what you might expect in the way of performance impact. This type of calculation most likely will not represent actual performance, but it can provide a bit of insight into fragmentation and its theoretical impact on a system.

Most of my partitions are around 1.5% or 1.6% fragmented; I do have one that is 3.3% fragmented but that is a large, 128GB filesystem with fewer than 100 very large ISO image files; I've had to expand the partition several times over the years as it got too full.

That is not to say that some application environments don't require greater assurance of even less fragmentation. The EXT filesystem can be tuned with care by a knowledgeable admin who can adjust the parameters to compensate for specific workload types. This can be done when the filesystem is created or

later using the **tune2fs** command. The results of each tuning change should be tested, meticulously recorded, and analyzed to ensure optimum performance for the target environment. In the worst case, where performance cannot be improved to desired levels, other filesystem types are available that may be more suitable for a particular workload. And remember that it is common to mix filesystem types on a single host system to match the load placed on each filesystem.

Due to the low amount of fragmentation on most EXT filesystems, it is not necessary to defragment. In any event, there is no safe defragmentation tool for EXT filesystems. There are a few tools that allow you to check the fragmentation of an individual file or the fragmentation of the remaining free space in a filesystem. There is one tool, **e4defrag**, which will defragment a file, directory, or filesystem as much as the remaining free space will allow. As its name implies, it only works on files in an EXT4 filesystem, and it does have some limitations.

If it becomes necessary to perform a complete defragmentation on an EXT filesystem, there is only one method that will work reliably. You must move all the files from the filesystem to be defragmented,

ensuring that they are deleted after being safely copied to another location. If possible, you could then increase the size of the filesystem to help reduce future fragmentation. Then copy the files back onto the target filesystem. Even this does not guarantee that all the files will be completely defragmented.

Conclusions

The EXT filesystems have been the default for many Linux distributions for more than 20 years. They offer stability, high capacity, reliability, and performance while requiring minimal maintenance. I have tried other filesystems but always return to EXT. Every place I have worked with Linux has used the EXT filesystems and found them suitable for all the mainstream loads used on them. Without a doubt, the EXT4 filesystem should be used for most Linux systems unless there is a compelling reason to use another filesystem.