Red Hat's New Journaling File System: ext3

Abstract

In Red Hat Linux 7.2, Red Hat provides its first officially supported journaling file system: ext3. The ext3 file system is a set of incremental enhancements to the robust ext2 file system that provide several advantages. This paper summarizes some of those advantages (first in general terms and then more specifically), explains what Red Hat has done to test the ext3 file system, and (for advanced users only) touches on tuning.

Other journaling file systems are being written for Linux; this paper does not address them in any way and is not intended to compare ext3 to any other journaling file system.

Table of Contents

What are the advantages of ext3?

Why do you want to migrate from ext2 to ext3? Four main reasons: availability, data integrity, speed, and easy transition.

Availability

After an unclean system shutdown (unexpected power failure, system crash), each ext2 file system cannot be mounted until its consistency has been checked by the e2fsck program. The amount of time that the e2fsck program takes is determined primarily by the size of the file system, and for today's relatively large (many tens of gigabytes) file systems, this takes a long time. Also, the more files you have on the file system, the

longer the consistency check takes. File systems that are several hundreds of gigabytes in size may take an hour or more to check. This severely limits availability.

By contrast, ext3 does not require a file system check, even after an unclean system shutdown, except for certain rare hardware failure cases (e.g. hard drive failures). This is because the data is written to disk in such a way that the file system is always consistent. The time to recover an ext3 file system after an unclean system shutdown does not depend on the size of the file system or the number of files; rather, it depends on the size of the "journal" used to maintain consistency. The default journal size takes about a second to recover (depending on the speed of the hardware).

Data Integrity

Using the ext3 file system can provide stronger guarantees about data integrity in case of an unclean system shutdown. You choose the type and level of protection that your data receives. You can choose to keep the file system consistent, but allow for damage to data on the file system in the case of unclean system shutdown; this can give a modest speed up under some but not all circumstances. Alternatively, you can choose to ensure that the data is consistent with the state of the file system; this means that you will never see garbage data in recently-written files after a crash. The safe choice, keeping the data consistent with the state of the file system, is the default.

Speed

Despite writing some data more than once, ext3 is often faster (higher throughput) than ext2 because ext3's journaling optimizes hard drive head motion. You can choose from three journaling modes to optimize speed, optionally choosing to trade off some data integrity.

- One mode, `data=writeback`, limits the data integrity guarantees, allowing old data to show up in files after a crash, for a potential increase in speed under some circumstances. (This mode, which is the default journaling mode for most journaling file systems, essentially provides the more limited data integrity guarantees of the ext2 file system and merely avoids the long file system check at boot time.)
- The second mode, `data=ordered` (the default mode), guarantees that the data is consistent with the file system; recently-written files will never show up with garbage contents after a crash.
- The last mode, `data=journal`, requires a larger journal for reasonable speed in most cases and therefore takes longer to recover in case of unclean shutdown, but is sometimes faster for certain database operations.

The default mode is recommended for general-purpose computing needs. To change the mode, add the `data=`*`something`* option to the mount options for that file system in the `/etc/fstab` file, as documented in the mount man page (**man mount**).

Easy Transition

It is easy to change from ext2 to ext3 and gain the benefits of a robust journaling file system, without reformatting. That's right, there is no need to do a long, tedious, and error-prone backup-reformat-restore operation in order to experience the advantages of ext3. There are two ways to perform the transition:

- The Red Hat Linux installation program offers to transition your file systems when you upgrade your system. All you have to do is select one checkbox per file system.
- The `tune2fs` program can add a journal to an existing ext2 file system. If the file system is already mounted while it is being transitioned, the journal will be visible as the file `.journal` in the `root` directory of the file system. If the file system is not mounted, the journal will be hidden and will not appear in the file system. Just run

  ```
  tune2fs -j /dev/hda1
  ```

- (or whatever device holds the file system you are transitioning) and change **`ext2`** to **`ext3`** on the matching lines in `/etc/fstab`. If you are transitioning your root file system, you will have to use an initrd to boot. Run the `mkinitrd` program as described in the manual and make sure that your LILO or GRUB configuration loads the initrd. (If you fail to make that change, the system will still boot, but the root file system will be mounted as ext2 instead of ext3 — you can tell this by looking at the output of the command **cat /proc/mounts**.) More information on `tune2fs` can be found in the `tune2fs` man page (**man tune2fs**).

Why ext3?

A list of reasons why Red Hat chose ext3 for our first officially supported journaling file system follows. Note that these reasons are not necessarily each unique to ext3 (some other journaling file systems share several of the points here), but the whole set of reasons taken together is unique to ext3.

- ext3 is forward and backward compatible with ext2, allowing users to keep existing file systems while very simply adding journaling capability. Any user who wishes to un-journal a file system can do so easily (not that we expect many to do so...). Furthermore, an ext3 file system can be mounted as ext2 without even removing the journal, as long as a recent version of `e2fsprogs` (such as the one included in Red Hat Linux 7.2) is installed.
- ext3 benefits from the long history of fixes and enhancements to the ext2 file system, and will continue to do so. This means that ext3 shares ext2's well-known robustness, but also that as new features are added to ext2, they can be carried over to ext3 with little difficulty. When, for example, extended attributes or

HTrees are added to ext2, it will be relatively easy to add them to ext3. (The extended attributes feature will enable things like access control lists; `HTrees` make directory operations extremely fast and highly scalable to very large directories.)

- ext3, like ext2, has a multi-vendor team of developers who develop it and understand it well; its development does not depend on any one person or organization.
- ext3 provides and makes use of a generic journaling layer (jbd) which can be used in other contexts. ext3 can journal not only within the file system, but also to other devices, so as NVRAM devices become available and supported under Linux, ext3 will be able to support them.
- ext3 has multiple journaling modes. It can journal all file data and metadata (`data=journal`), or it can journal metadata but not file data (`data=ordered` or `data=writeback`). When not journaling file data, you can choose to write file system data before metadata (`data=ordered`; causes all metadata to point to valid data), or not to handle file data specially at all (`data=writeback`; file system will be consistent, but old data may appear in files after an unclean system shutdown). This gives the administrator the power to make the trade off between speed and file data consistency, and to tune speed for specialized usage patterns.
- ext3 has broad cross-platform compatibility, working on 32- and 64- bit architectures, and on both little-endian and big-endian systems. Any system (currently including many Unix clones and variants, BeOS, and Windows) capable of accessing files on an ext2 file system will also be able to access files on an ext3 file system.
- ext3 does not require extensive core kernel changes and requires no new system calls, thus presenting Linus Torvalds no challenges that would effecitvely prevent him from integrating ext3 into his official Linux kernel releases. ext3 is already integrated into Alan Cox's `-ac` kernels, slated for migration to Linus's official kernel soon.
- The e2fsck file system recovery program has a long and proven track record of successful data recovery when software or hardware faults corrupt a file system. ext3 uses this same e2fsck code for salvaging the file system after such corruption, and therefore it has the same robustness against catastrophic data loss as ext2 in the presence of data-corruption faults.

Again, we do not claim that each one of these points is unique to ext3. Most of them are shared by at least one other file system. We merely claim that the set of all of them together is true only for ext3. The expressed needs of our customers drove our decisions about what feature set was important for us to support right now. In our judgement, ext3 currently has the best fit for our customers' requirements. We will continue to evaluate other file systems for inclusion in future versions of Red Hat Linux.

Why can you trust ext3?

Here are some of the things Red Hat has done to ensure that ext3 is safe for hadling user data.

- We have performed extensive stress testing under a large set of configurations. This has involved many thousands of hours of "contrived" load testing on a wide variety of hardware and file system configurations, as well as many use case tests.
- We have audited ext3 for multiple conditions, including memory allocation errors happening at any point. We test that repeatedly and often (every time the code changes) by forcing false errors and testing file system consistency.
- We audited and tested ext3 for poor interactions with the VM subsystem, finding and fixing several interactions. A journaling file system puts more stress on the VM subsystem, and we found and fixed bugs both in ext3 and in the VM subsystem in the process of this auditing and testing. After thousands of hours of this testing, we are extremely confident in the robustness of the ext3 file system.
- We have done an extensive year-long-plus beta program, starting with ext3 on the 2.2 kernel series, and then moving forward to the 2.4 kernel series. Even before the official beta program, ext3 was put into production use in some circumstances; ext3 has been in production use on some widely-accessed servers, including the rpmfind.net servers, for more than two years.
- We have arranged to allow the user to choose to check file systems consistency after unclean system shutdown, even if the filesystem is marked "clean", in order to deal with potential hardware-generated corruption. This is because hardware failures, and most particularly real power failures or brownouts, can cause "garbage" data to be written practically anywhere on disk. Hitting the reset button is not likely to trigger this kind of problem, but a true power failure associated with things like lightning strikes and trees falling on power lines tend to involve spikes and brownouts that can damage date en route to disk. IDE systems tend to be a bit more susceptible to this kind of problem than SCSI systems, in part because IDE disks tend to implement looser cacheing algorithms.

This feature is implemented using the `/.autofsck` file — if the root user removes that file during normal operation, the system will offer the choice to check file system consistency at boot time. If `/.autofsck` is missing and the user elects to force file system consistency checks, the effect will be the same as if the `/forcefsck` file existed.

Tuning suggestions

Choosing elevator settings

The ext3 file system acts a bit differently than the ext2 file system, and the differences can appear in various ways. Advanced users may choose to tune the file system and I/O system for performance. This is an introduction to some of the more common tuning that advanced users may wish to try. All tuning, of course, needs to be done in the context of performance testing of specific applications; there is no "one size fits all" approach to tuning. This is, however, intended to provide some generally useful information.

Most Linux block device drivers use a generic tunable "elevator" algorithm for scheduling block I/O. The `/sbin/elvtune` program can be used to trade off between throughput and latency. Given similar loads, the ext3 file system may require smaller latency numbers as provided to the `/sbin/elvtune` program in order to provide similar results to the ext2 file system.

In some cases, attempting to tune for maximum throughput at the expense of latency (in this case, large read latency (`-r`) and write latency (`-w`) numbers used with the `/sbin/elvtune` program) can actually decrease throughput while increasing latency. This effect is more pronounced with the ext3 file system for a variety of reasons.

- With the ext2 file system, writes are scheduled every 30 seconds; with the ext3 file system, writes are scheduled every 5 seconds. This keeps journal transactions from having a noticeable impact on system throughput and also keeps data on disk more up-to-date.
- The ext3 file system, by journaling all metadata changes, can magnify the effect of atime changes significantly. You can mount a file system with the `noatime` flag in order to disable atime updates. While this is not the only source of metadata updates, on many systems, particularly highly-accessed servers with lots of accessed files, atime updates can be responsible for the majority of metadata updates, and on these systems, turning off atime updates may noticeably reduce latency and increase throughput.

In order to tune for our default file system choice of ext3, Red Hat has reduced the default read and write latency numbers to half of the default values (from 8192 read, 16384 write to 4096 read, 8192 write). We expect that in general use, you will not have to change these numbers; we hope we have already done this for you. Our changed default values have produced good results in our tests. However, in order to tune for specific applications, we suggest benchmarking your applications with a variety of values, testing interactive response during some runs if interactive response is important to you. In general, we recommend that you set read latency (`-r`) to half of write latency (`-w`).

For example, you might run:

```
/sbin/elvtune -r 1024 -w 2048 /dev/sdd
```

to change the elevator settings for the device `/dev/sdd` (including all the partitions on `/dev/sdd`). Changes to the elevator settings for a partition will apply to the elevator for the device the partition is on; all partitions on a device share the same elevator.

Once you have found elvtune settings that give you your most satisfactory mix of latency and throughput for your application set, you can add the calls to the `/sbin/elvtune` program to the end of your `/etc/rc.d/rc.local` script so that they are set again to your chosen values at every boot.

Choosing journaling mode

Speed

There are some characteristic loads that show very significant speed improvement with the `data=writeback` option, which provides lower data consistency guarantees. In those cases, the data consistency guarantees are essentially the same as the ext2 file system; the difference is that the file system integrity is maintained continuously during normal operation (this is the journaling mode used by most other journaling file systems). One of these cases involves heavy syncronous writes. Other cases involve creating and deleting large numbers of small files heavily, such as delivering a very large flow of small email messages. If you switch from ext2 to ext3 and find that your application performance drops substantially, the `data=writeback` option is likely to give you a significant amount of performance back; you will still have some of the availability benefits of ext3 (file system is always consistent) even if you do not have the more expensive data consistency guarantees.

Red Hat is continuing to work on several performance enhancements to ext3, so you can expect several of these cases to improve in the future. This means that if you choose `data=writeback` now, you may want to retest the default `data=journal` with future releases to see what changes have been made relative to your workload.

Data integrity

In most cases, users write data by extending off the end of a file. Only in a few cases (such as databases) do users ever write into the middle of an existing file. Even overwriting an existing file is done by first truncating the file and then extending it again.

If the system crashes during such an extend in `data=ordered` mode, then the data blocks may have been partially written, but the extend will not have been, so the incompletely-written data blocks will not be part of any file.

The only way to get mis-ordered data blocks in `data=ordered` mode after a crash is if a program was overwriting in the middle of an existing file at the time of the crash. In such a case there is no absolute guarantee about write ordering unless the program uses `fsync()` or `O_SYNC` to force writes in a particular order.

In `data=journal` mode, even mid-file overwrites will be strictly ordered after a crash.