New Journaling filesystem with atomic transactions

Reiser's 4th Symphony

The newly developed Reiser 4 is rapidly approaching its premiere in the kernel 2.6. It promises atomic transactions and faster reading speeds than its predecessor ReiserFS; it makes better use of the hard disk and is extensible by means of plug-ins. BY MARCEL HILZINGER

www.photocase.de

After a development period of over four years, the Reiser filesystem looks set for a major milestone jump to version 4 later this year. Hans Reiser's developer team has re-programmed the new Reiser 4 from scratch. The team is currently testing the performance and stability of the designated heir to ReiserFS, and removing any minor bugs that it discovers in the process.

Among other things, the new journaling filesystem is capable of atomic transactions, and extensible by means of plugins. The Namesys homepage [1] has a test version available for downloading (see also the "Installation Guide" box, on the following page).

THE AUTHOR

After finishing studies in Zürich in 2000, Marcel Hilzinger has been working for Suse Linux's Hungarian office in Budapest, where, among other things, he translated the Suse documentation into Hungarian.

Basic Design

Put simply, the Reiser 4 filesystem consists of two layers: the storage layer, and the semantic layer. Objects exist in both layers. Reiser 4 does not distinguish between files and directories, but treats both as objects. Each object on the filesystem has an object ID and a key, which is partially generated by reference to the object ID.

As the name suggests, keys provide access to the filesystem. Reiser 4 uses keys to locate any of the objects. You can even use the key to locate a specific sector of a file. In the basic setup, Reiser 4 now uses a long key system (for example *10001:1:746f6d20776169:0:10278:0*) that comprises the elements shown in the following list:

- Major Locality
- Minor Locality
- Ordering
- Unused (always 0)
- Object ID
- Offset

While the storage layer is responsible for storing and organizing the data in the tree structure, the semantic layer is responsible for mapping filenames to keys. The filesystem can use keys to locate the required bytes in the storage layer. The offset provides the byte position within a given object.

Filesystem plugins

Reiser 4 uses plugins to handle file operations, using its own internal file and directory plugin to handle typical Unix file management tasks. Each file and each directory also has a plugin ID which describes a collection of outside methods. These methods can be used to tell the filesystem to execute outside operations with the file.

Outside operations are file management methods which are not part of the filesystem itself. They allow external programs to interact with integral parts of the filesystem without needing to compile a new kernel. Reiser 4 has over 30 internal plugins at present; you can enter the *fsck.Reiser 4 -l* command to view a list. There are no known external plugins at present. A quota plugin would be useful – Reiser 4 does not support quotas at present.

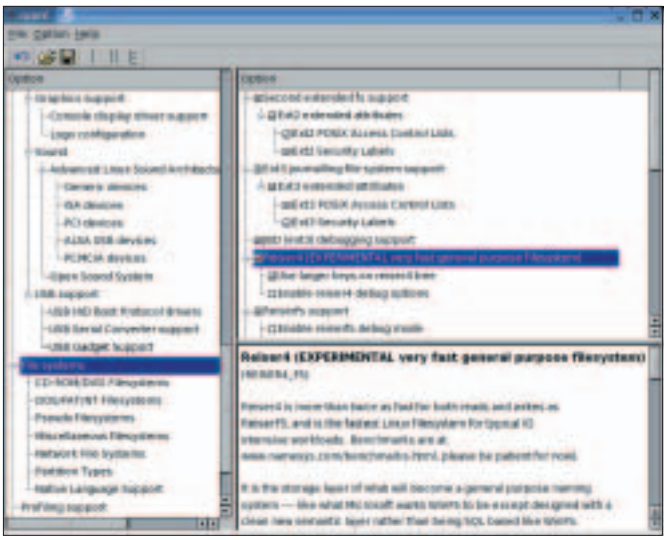

Figure 1: The patched kernel has an entry for Reiser 4 below *File systems*. The check mark means build the system into the kernel. Double clicking will change the check mark into a dot. Reiser 4 will also run as a kernel module.

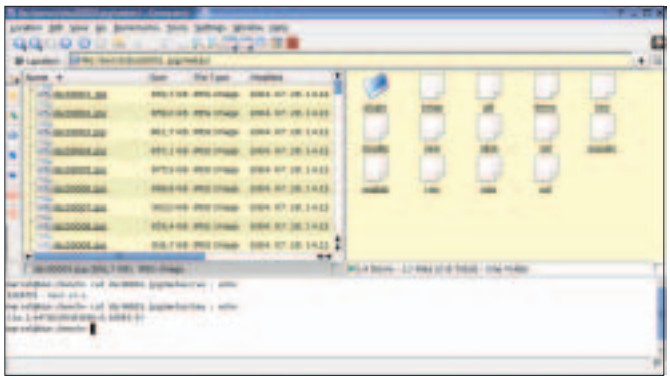

Figure 2: A directory with images on a Reiser 4 partition (left-hand side of window). Clicking will not take you to the pseudofiles. Adding *metas* to the path in the URL box, reveals the pseudofiles for the *dsc00001.jpg* image file, and the *plugin* directory (right).

The file and hash plugin is responsible for organizing objects. In contrast to most Linux filesystems, Reiser 4 does not organize files by creation date, but alphabetically. In the case of files with more than 15 characters in their names, the first eight characters are significant, followed by the hash of the whole filename.

Reiser 4 uses the hash plugin to organize directories. At present, it uses cookies made up of the hash value and a creation counter.

This is a temporary solution based on ReiserFS; in future, the file plugin will be responsible for putting directories in alphabetical order.

Security Modules

The security plugins also play an important role. This obviously has something to do with the Reiser 4's main sponsor, the Defense Advanced Research Projects Agency (DARPA), a research organization run by the US Department of Defense. One of the plugins handles large files that contain a lot of information.

Traditional filesystems apply access controls only to individual files. You can use access control lists (ACLs) to assign different privileges for the same file to different users, but if a user has been granted access, this privilege applies to the whole file.

Reiser 4 solves this problem by dividing a file up into a collection of items; for example, */etc/passwd* would be divided by reference to its individual lines. Each item can have privileges, and use different plugins. Reiser 4 also allows you to assign a different delimiter for the item borders depending on the application. Applications still view individual items as files.

Installation Guide

This guide assumes the kernel 2.6.5 and the official Reiser 4 snapshots dated March 26, 2004 [2]. Be cautious with patches for more recent kernel versions [3], as error-free operations can not be guaranteed.

- Download the kernel sources off of the official server [4], and unpack the sources in */usr/src/*.
- Copy the *all.diff.gz* patch from the Namesys homepage to the directory for the new kernel */usr/src/linux-2.6.5/*.
- Change to the new kernel directory, and apply the patch using the *zcat all.diff.gz | patch -p1* command.
- There is an entry for *->host* missing in line 570 of *fs/reiser4/as_ops.c*. The line should read:

```
mapping->host->dirtied_when=
jiffies|1;
```

- Replace the entry for *mapping->dirtied _when = jiffies|1;* in line 477 of *fs/reiser4/plugin/tail.c* with the modified line:

```
minode->dirtied_when = jiffies|1;
```

- Working as root, launch the kernel configuration program, by typing *make xconfig* (you will need the QT 3 developer packages).
- Ensure that *Reiser4* is selected for *File systems*. You can opt for a static build or a dynamically loadable module, as shown in Figure 1.
- After saving the configuration, compile the kernel in the usual way by typing *make bzImage modules modules_install*, and then copy the new kernel to the target directory, by typing *cp arch/i386/boot/bzImage/boot/vmlinuz-reiser4* for example. Avoid overwriting the original kernel.
- If you use the Initial RAM Disk at boot time (like Suse), you will need to run *mkinitrd* to create a new RAM disk.
- To make it easier to boot the new kernel, add an entry to the Grub configuration file */boot/grub/menu.lst*.
- The */etc/fstab* file should not contain any distribution specific options that the default kernel can not handle. For example, Suse 9.1 had difficulty with the *acl* and *user _xattr* options. If you are unsure, remember that all you really need is an entry for *rw,exec*.
- You can now boot the new kernel. After doing so, you need to install *libaal-0.5.0.tar.gz* and *reiser4progs-0.5.3.tar.gz*. Follow standard procedure to do so: *./configure && make && make install*. If Reiser4progs fails to find the Libaal libraries after you have installed them, run *ldconfig* to remedy the problem.
- You can then run */usr/local/sbin/mkfs.reiser4 Partition* to create Reiser 4 partitions, and *mount* to mount them.

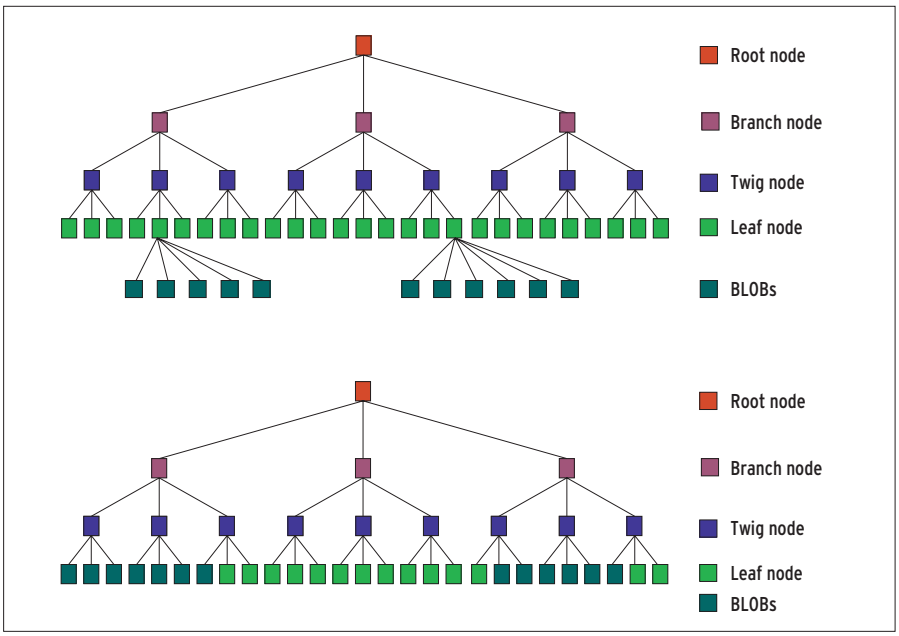

Figure 3: A traditional balanced tree (top), as used by ReiserFS 3.6 has a special approach to storing BLOBs, using a leaf to assume the role of an internal node. In contrast, a Reiser 4 tree (bottom) stores BLOBs in the leaf node (extents). This keeps the number of internal nodes to a minimum.

Files and Subdirectories

As previously mentioned, Reiser 4 is not an extension of ReiserFS, but a completely new development. The designer, Hans Reiser, insisted on keeping the code for the new filesystem as simple as possible. The "ReiserFS – More than a theory" box provides details on Reiser's beginnings.

Reiser avoids overloading the filesystem with attributes, such as the extended attributes found in Ext 2. The idea is to achieve functionality by as simple an approach as possible, using files. To this end, every object on a Reiser 4 filesystem has a kind of sub-namespace in the *metas* directory, which the *readdir* system call will not list, but which you can change to by typing *cd metas*. This also holds good for files: *cd filename/metas* will change to the metas directory for a file. However, the file, itself, has to be executable for this feature to work.

The pseudofiles (see Figure 2, previous page) in the *metas* directory are objects attached to other objects (files or directories). They do not really have a separate existence on the disk (in contrast to the files below */proc* or */sys*). Reiser 4 uses them to implement various standard Unix functions, as well as any special security or compression methods.

For example, you can change the owner of a file without calling *chown*, or use a simple *cat* command to view the access privileges for a directory (Figure 2 bottom). Table 1 provides a short description of the major pseudofiles in *metas*.

Living Filesystem

Modern Unix filesystems like ReiserFS, XFS, and JFS are all based on tree structures. We need to distinguish between B+ and B+ tree variants at this point.

Table 1: Important Pseudofiles

File/Directory	Description	Explanation
bmap	List of block numbers assigned to an object	One block number per line
gid	ID of an object's primary group	Example: 500
items	List of elements that form the object	Example: *(2a:4:666f6f31000000:0:6d352:2) body length: 3*
key	Object key	Example: *2a:4:666f6f31000000:0:6d352:0*
locality	Major Locality of object	The same for all objects within a directory
new	Pseudofile for creating new objects	
nlink	Number of hard links to an object	
oid	Object ID of object	Example: *97549*
pagecache	Cache statistics of object	You can read the file en bloc, or use the sub-objects *clean*, *dirty*, *io*, *locked* and *nrpages* to read it as a directory
plugin	Plugins available for an object.	You can type *mkfs.Reiser4 -l* to output the list
pseudo	List of pseudofiles for an object	
readdir	List of sub-objects assigned to an object.	In case of directories, lists the directory contents
rwx	Access controls for an object in numerical form and clear text	Example: *0100755 -rwxr-xr-x*
size	Size of object in bytes	A new empty directory on Reiser 4 occupies just 2 bytes plus 1 byte for each new sub-object
uid	ID of object owner	Example: *500*

ReiserFS – More than a theory

Hans Reiser worked on his theory of databases and namespaces for over eight years until, influenced by the theory of "Roads and Waterways", he finally reached the conclusion that the filesystem has the same significance for the computer, as roads and waterways have for our civilization. The better-connected a system is, the better its ability to interact becomes, and the better it can advance.

Reiser is not primarily a developer who wants to create a filesystem, but a theoretician who regards a filesystem as the perfect application of his theory. As Reiser wanted to create his filesystem for and using Linux right from the outset, and as funding was limited, he looked for affordable and qualified programmers. Thus, development work on ReiserFS started in 1993 with Hans Reiser and a team of young Russian developers.

ReiserFS 3.6 has been classified as stable since the kernel 2.4.21. Namesys, the company behind developing Reiser 4, put the new version through intensive crash tests in 2003. It was released for public testing in 2004 after removing the known bugs.

Whereas B+ trees only store pointers and administrative information in their internal nodes, B- trees use the nodes to store data. As B+ stores data in the leaves of the tree only, B+ needs more leaves than B- to store the same amount of data. This said, B+ manages the internal nodes more efficiently as they only contain pointers.

ReiserFS and Reiser 4 both use B+ trees. The major difference between traditional balanced trees, such as those used by ReiserFS 3.6, and the new dancing trees, as used by Reiser 4, is the way they handle BLOBs (Binary Large OBjects) and internal nodes.

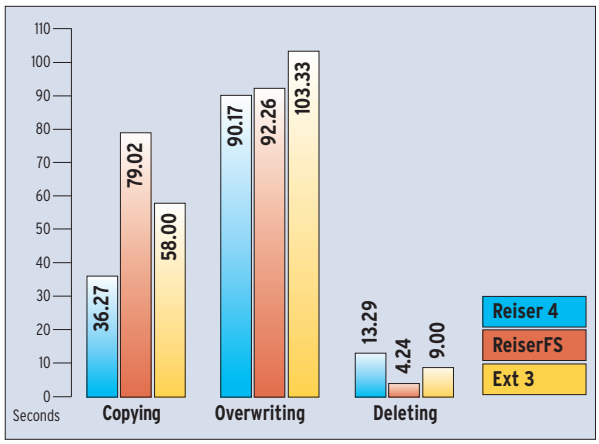

Figure 4: The first write operation with the kernel sources shows that Reiser 4 is almost twice as quick as Ext 3. The candidates are more evenly matched on overwrite operations. ReiserFS deletes more quickly than any others.

BLOBs are files that need more space than an end-node (leaf) can provide. To store BLOBs, traditional databases and ReiserFS use a leaf in one node to store pointers to other leaves. This creates additional internal nodes, and the Highly Balanced Tree (where the path from any leaf to the root should be the same length) becomes unbalanced, see Figure 3, top.

Reiser 4 store BLOBs directly in the leaves, treating BLOBs just like any other data (see Figure 3, bottom). This minimizes the number of internal nodes, and keeps the tree balanced. On his homepage, Reiser states that the source code for the kernel 2.4.18 occupies 1629 internal nodes on ReiserFS, in comparison to a mere 164 on Reiser 4.

On an average computer with Reiser 4, all the internal nodes of the filesystem tree should easily fit into memory. If it can not avoid swapping out to disk, Reiser 4 first pushes all the nodes as far left as they will go in the tree, and then gets more nodes from the free space on the right.

Dancing Trees

Continually balancing the tree would add unnecessarily complexity with all the data in main memory, but it does makes sense to save disk space, and thus avoid I/O operations. To do this, Reiser 4 compresses the data stored in memory immediately before writing to disk. This approach improves performance. Traditional balanced trees, such as those use by

ReiserFS, are typically more compact, but they need to shift data around more frequently. The loss of compactness is one of the drawbacks of the dancing trees used by Reiser 4. A repacker planned for version 4.1 will help to mitigate the impact.

ReiserFS assigns new nodes when creating a block number, whereas Reiser 4 waits until the system writes the data in main memory out to disk. This feature, which was modeled on XFS, means that new files that are deleted without being stored have no impact on the filesystem, which in turn means a performance boost.

Atomic Transaction

Atomic transactions are a traditional requirement for filesystems. Let's assume that a power failure occurs while a user is moving a file from A to B. On an atomic filesystem, the file is either at location A or at location B after power is restored, but under no circumstances is the file partly at A and partly at B. The current approach to handling this is to run the *fsck* program to check the filesystem after a failure, and save any fragments it finds in the *lost+found* directory.

Reiser 4 has an atomic approach to internal file moves. Although file fragments may exist after a system failure during copying, the fragments will contain data from the copied file only, rather than binary junk. This is important for security reasons, as scrap files can contain personal information and this should obviously be avoided.

Depending on the circumstances, Reiser 4 will write files once (from the source to the target) or twice (from the source to the journal, and from the journal to the target). To provide for atomic operations with any single write transactions, Reiser 4 simply swaps the journal for the target.

If Reiser 4 needs to copy a file from directory A to B, it will first write the file to LOG, that is the storage location of the journal, and then point the nodes

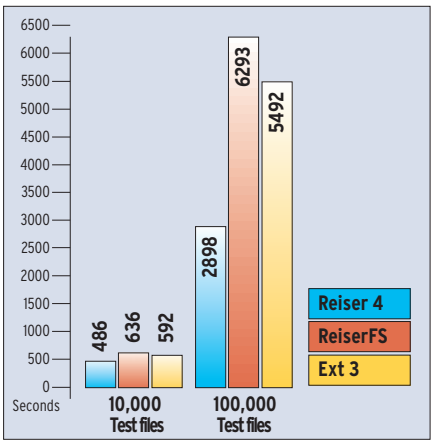

Figure 5a: The more files the 1 GByte of data include, the quicker Reiser 4 performs the Bonnie++ benchmark in comparison to other filesystems.

that pointed at B up to this point at LOG. The newly-created journal thus wanders over to a new position, which thus explains the name of a wandering log.

During a transaction, the Reiser 4 system keeps modified blocks where they are until the system confirms that the transaction has been completed. After receiving this message, Reiser 4 goes on to complete the transaction; the number of write operations depends on the changes.

In case of minor changes to large files, it makes sense to use two write operations. This means that the file can stay put while the changes are written. In case of major changes, it makes more sense to change the node, and leave the rest of the work to the repacker. This feature is not implemented at present, as the repacker does not yet exist. As a result, Reiser 4 will always use two write operations.

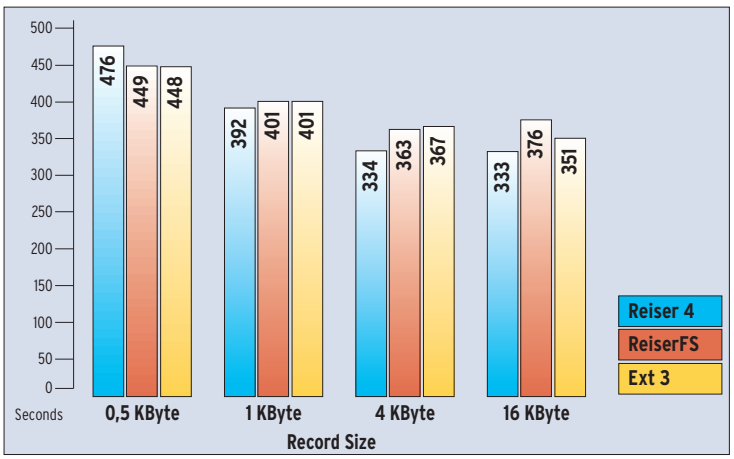

Figure 5b: IOzone uses a 1 GByte file for its benchmark. Reiser 4 is slower than ReiserFS or Ext 3 with the smallest records, but sizes of 4 or 16 KBytes are more relevant in practical applications.

Performance

As it is a multi-purpose filesystem, we put Reiser 4 through various benchmarks on a normal desktop PC. Our lab machine has 256MBytes RAM and a 40GByte hard disk (Maxtor Baracuda, ATA133). We used a copy of Suse Professional Linux 9.1 with the kernel 2.6.7-mm4 as the operating system, and applied a recent Reiser 4 patch, dated July 6, 2004 [3].

We used the *fs_bench* from [5] as our filesystem benchmark. *fs_bench* combines the Bonnie++ [6] and IOzone [7] benchmarks. Besides this test, we also used the results of the *slow.c* benchmark from the Namesys homepage and our own benchmarking tests.

Reiser 4 lives up to all of its hype with excellent results, particularly when handling small files where it has the best benchmark results by far (see Figure 4, previous page). The new filesystem is not only quick, it also saves disk space, with a kernel 2.6.7 build occupying a mere 371 MBytes of disk space under Reiser 4. ReiserFS needs 431 MBytes and Ext 3 an enormous 446 MBytes of hard disk space to store the same amount of data.

One thing we noticed was that the hard disk was a lot quieter during the Reiser 4 test than it was with ReiserFS or Ext 3. The Namesys developers really seem to have been able to put the results of their hard disk head movement tests to good use, optimizing Reiser 4 for write operations [8].

Reiser, the Turbo Copier

As you can see in Figure 4, Reiser 4 wins hands down in the kernel source code copy test. Our three test candidates showed similar performance in the overwrite test, although Reiser 4 is slightly quicker than its competitors.

However, ReiserFS deletes quicker than the other candidates; Reiser 4 has lost a lot of ground in this discipline. This can be

attributed to the filesystem's internal structure, and turns out to be an advantage when deleting files of GByte magnitudes: Reiser 4 needs just 1.5 seconds to delete a 6 GByte file, whereas ReiserFS takes 12 seconds, with Ext 3 losing out at 14 seconds.

The Bonnie++ and IOzone benchmarks return a mixed bunch of results. Reiser 4 is extremely quick in many disciplines, but there are still a few places where it loses out to the previous ReiserFS. An example of this would be deleting in the random create and sequential create tests with Bonnie++. Reiser 4 continues to produce heavy CPU load; most of the test results do not compare well with Ext 3, and even with ReiserFS at times.

The aggregate time taken to complete the benchmarks is a good measure of how well-balanced a filesystem is. You can check the results in Figures 5a and 5b on the previous page. Based on total time for the tests, Reiser 4 is the obvious winner.

Seriously Quick

The *slow.c* benchmark basically demonstrates that Reiser 4 performs better than ReiserFS when required to write multiple data streams at the same time (see Figure 6, right). In comparison to writing or reading a single file, multi-stream operations really impact ReiserFS performance-wise. In contrast, Reiser 4's performance is more or less unchanged for this operation.

Ext 3 can keep pace with Reiser 4 for write operations, but really loses ground in operations with four parallel streams. Reiser 4 gets amazingly close to the hard disk's max transfer rate of 27.6 MBytes (according to *hdparm -t*). In future, the new filesystem is expected to break records using a compression plugin, and achieve higher transfer rates than the max physical transfer speed of the drive.

There are not many benchmarks that test the time required to mount and create a filesystem. This can be the vital statistic which prompts you to opt for a specific filesystem in a production environment. Reiser 4 still has some way to go as regards mount times, taking around 12 seconds for a 200 GByte partition. In comparison, ReiserFS takes 4 seconds, and Ext 3 less than 1 second to mount the same partition. On a more positive note, Reiser 4 took just 1 second to create a partition, whereas ReiserFS needed 4.

This difference is attributable to the fact that Reiser 4 simply creates the nodes required by the filesystem at this point. The filesystem occupies only a minimum of disk space. In contrast to this, ReiserFS creates an area for logfiles, and occupies about 30 MBytes on the newly created partition from the outset.

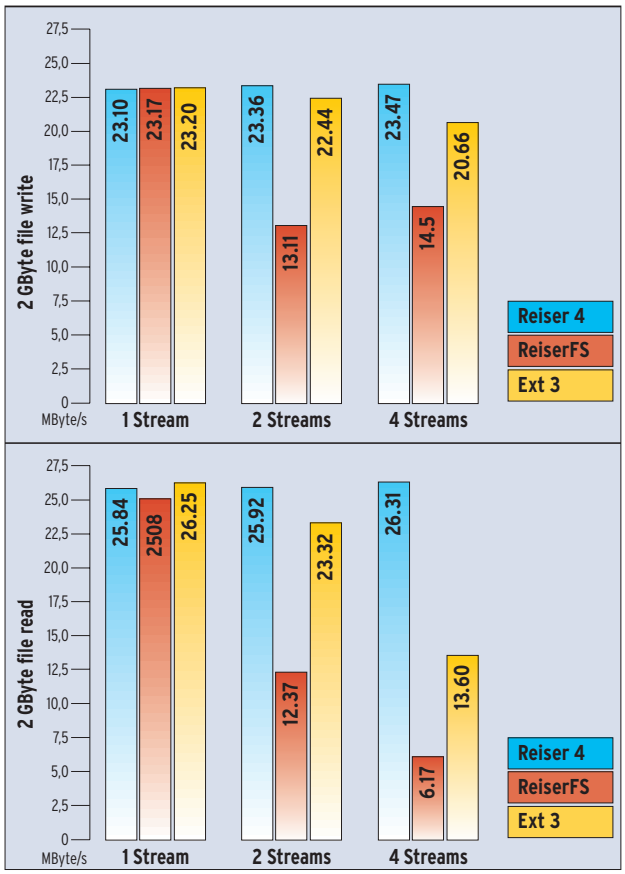

Figure 6: The *slow.c* benchmark tests the speed of the filesystem for parallel write operations with multiple data streams. There is very little sign of impact to Reiser 4's performance, Ext 3 shows some impact, and ReiserFS is noticeably slower.

Creating a 200 GByte Ext3 partition reminded us of the bad old days with DOS. Ext3 took no less than 5 minutes to create the Ext2 inode table and the resulting logfiles.

Finally, we put the Reiser 4 filesystem through several crash tests in the Linux labs. We pressed our lab system's reset button to cold boot the machine while Reiser 4 was busy with various operations. Although we repeated the test 15 times, Reiser 4 showed no sign of weaknesses; the files we tested were all readable, and the filesystem remained undamaged.

Reiser 4 has a whole bunch of new features and most benchmarks look really promising. The only question is whether Reiser 4 can live up to the expectations of the Linux Community. For almost two years, the Namesys homepage announced that Reiser 4 would be released "this summer". Now, after completing internal crash testing, the filesystem needs a lot of external testing and feedback from the Community.

Linspire has already announced that it will be integrating Reiser 4 with its distribution as soon as possible, and Suse is also interested in including Reiser 4 as soon as it reaches an acceptable level of stability.

Christmas Box

Currently, Namesys is looking for sponsors, as the DARPA funds can only be invested in the development of security features. So if you are looking forward to installing Reiser 4 on your computer this Christmas, you might like to consider a donation to Namesys. ■

INFO

[1] Reiser: *http://www.namesys.com*

[2] Snapshot of Kernel 2.6.5: *http://www.namesys.com/snapshots/LATEST/*

[3] Patches for current kernel: *http://www.namesys.com/auto-snapshots/*

[4] Linux kernel: *http://www.kernel.org*

[5] Fsbench: *http://fsbench.netnation.com*

[6] Bonnie++: *http://www.coker.com.au/bonnie++/*

[7] IOzone: *http://www.iozone.org*

[8] Reiser 4 design: *http://www.namesys.com/v4/reiser4_the_atomic_fs.html*