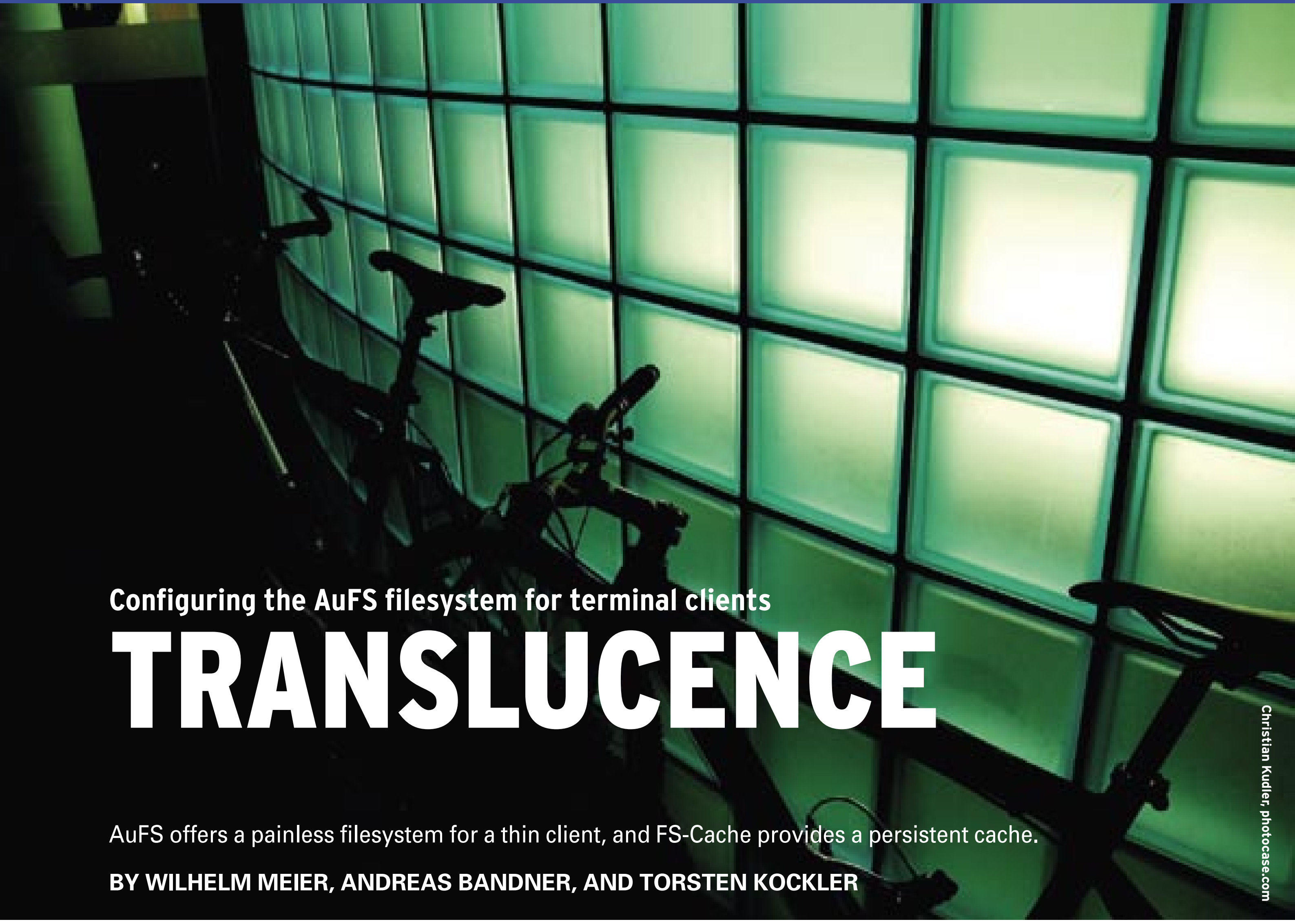


Configuring the AuFS filesystem for terminal clients

TRANSLUCENCE

AuFS offers a painless filesystem for a thin client, and FS-Cache provides a persistent cache.

BY WILHELM MEIER, ANDREAS BANDNER, AND TORSTEN KOCKLER

A thin client is often an attractive option for server-based computing environments. Thin clients save on hardware and administrative overhead by minimizing client configuration and concentrating resources on the server side. The very versatile Linux is an excellent option for a thin-client operating system; however, because one of the primary reasons for a thin client is to save administration time, it is important for the thin client to be as close as possible to an ordinary Linux computer. The root filesystem configuration, for instance, should resemble as closely as possible the configuration for a normal Linux system.

One way to reach this goal is by the use of a translucent filesystem, which combines the contents of separate filesystem *branches* into a single virtual filesystem. In the case of a diskless client, one branch could be an NFS-based directory located on the server, and the other branch could be a local filesystem stored on a RAM disk. The NFS server exports the root directory to all clients for read-only access. Local file modifications in */etc*, */var*, or */media* are then copied to a RAM disk by the translucent filesystem.

Unionfs [1] is perhaps the most popular translucent filesystem; however, Unionfs builds on the FiST framework, and this adds a great deal of complexity. Although the performance effects are typically not too serious, Unionfs has other problems.

AuFS [2] is an alternative translucent filesystem. Knoppix version 5.1 and newer dropped Unionfs in favor of AuFS, which was programmed from scratch and does not share its code base with Unionfs, although it provides similar functionality. AuFS stability and performance are better than Unionfs, although Linux distributions have not gotten around to building binary packages.

In this article, we describe a method for configuring a Linux thin client with the use of AuFS and an NFS-based persistent cache for better performance.

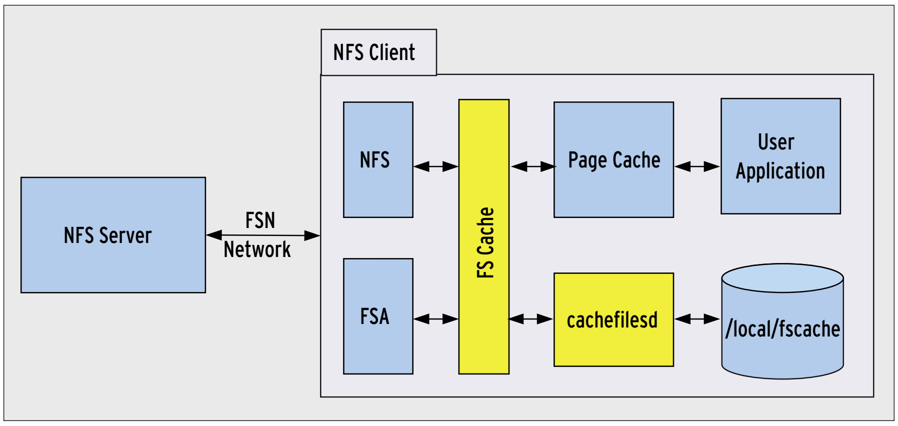
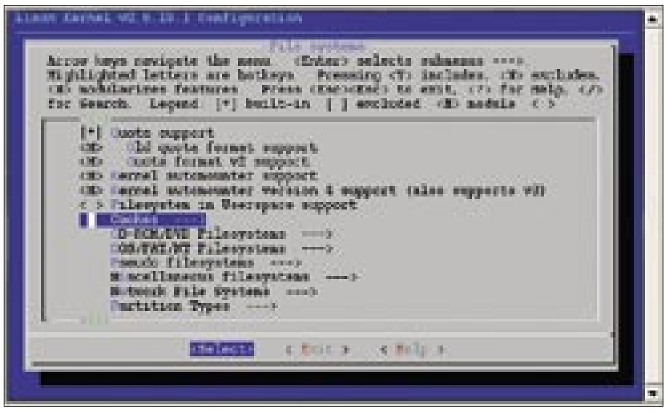


Figure 1: Interaction between FS-Cache components.

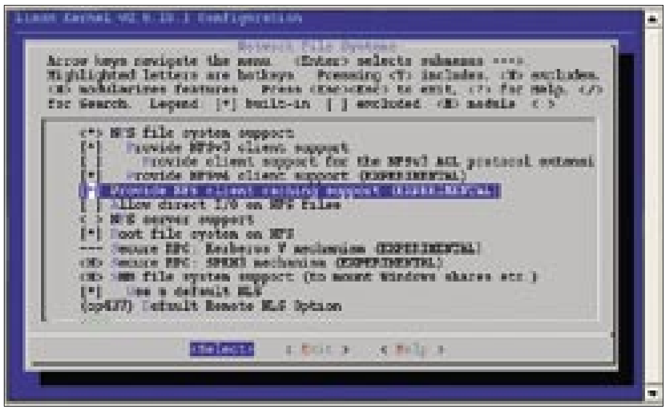


```
Linux Kernel v2.6.19.1 Configuration
                              File systems
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      [*] Quota support
      <M>   Old quota format support
      <M>   Quota format v2 support
      <M> Kernel automounter support
      <M> Kernel automounter version 4 support (also supports v3)
      < > Filesystem in Userspace support
          Caches  --->
          CD-ROM/DVD Filesystems  --->
          DOS/FAT/NT Filesystems  --->
          Pseudo filesystems  --->
          Miscellaneous filesystems  --->
          Network File Systems  --->
          Partition Types  --->

                <Select>    < Exit >    < Help >
```

Figure 2: Build in all the options in the Caches menu item.



```
Linux Kernel v2.6.19.1 Configuration
                          Network File Systems
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      <*> NFS file system support
      [*]   Provide NFSv3 client support
      [ ]     Provide client support for the NFSv3 ACL protocol extension
      [*]   Provide NFSv4 client support (EXPERIMENTAL)
      [*]   Provide NFS client caching support (EXPERIMENTAL)
      [ ]   Allow direct I/O on NFS files
      < > NFS server support
      [*] Root file system on NFS
      --- Secure RPC: Kerberos V mechanism (EXPERIMENTAL)
      <M> Secure RPC: SPKM3 mechanism (EXPERIMENTAL)
      <M> SMB file system support (to mount Windows shares etc.)
      [*]   Use a default NLS
      (cp437) Default Remote NLS Option

                <Select>    < Exit >    < Help >
```

Figure 3: The highlighted options enable the cache for NFS.

The FS-Cache [3] project supports the establishment of a persistent cache for network filesystems (Figure 1). FS-Cache, which only works for AFS and NFS right now, comprises multiple components, starting with a couple of patches for the Linux kernel that stand a good chance of making their way into the official kernel in the near future. Also included is a cache back-end and a modified *mount* program that understands the *fsc* option, which announces the use of a cache later.

The cache back-end has two options: *cachefilesd* and *cachefs*. The *cachefilesd* option uses an *ext3* filesystem as its persistence layer, whereas *cachefs* uses a block device as the cache. The *cachefs* algorithm tends to fragment the block device over time, which means major performance hits. The developers themselves advise users not to opt for *cachefs*.

Integrating AuFS

AuFS includes a kernel module and user-space tools. Kernel versions as of 2.6.19 additionally require an uncritical patch if one branch of the union resides on an NFS filesystem. Another uncritical patch exports a kernel function for reasons of efficiency. The command

```
cvs -d :pserver:anonymous@↩
aufs.sourceforge.net:↩
/cvsroot/aufs co aufs
```

identifies the current version in the *aufs* subdirectory. In the makefile, *local.mk*, you need to modify the following lines for the patches referred to earlier:

```
CONFIG_AUFS_LHASH_PATCH = y
...
CONFIG_AUFS_KSIZE_PATCH = y
```

The following patch commands update the kernel sources:

```
cd /usr/src/linux
patch -p0 < ↩
/root/aufs/lhash.patch
patch -p0 ↩
< /root/aufs/ksize.patch
```

Then, you can go on to install the kernel in the normal way. At the same time, you can create the *aufs* module:

```
cd /root/aufs
make KDIR=↩
/usr/src/linux -f local.mk
cp aufs.ko ↩
/lib/modules/2.6.19/fs/
depmod -a 2.6.19
```

After rebooting to the new kernel, you can use AuFS. The following command creates a union between the */tmp/readwrite* and */tmp/readonly* directories and mounts the union in */tmp/aufs*:

```
mount -t aufs -o ↩
dirs=/tmp/readwrite=↩
rw:/tmp/readonly=ro ↩
none /tmp/aufs
```

As you can see, AuFS is a direct replacement for any application for which you have used Unionfs with a diskless client.

Cache as Cache Can

Setting up FS-Cache is a slightly more involved process. Besides the kernel patches, you will need, as a minimum, a patched *mount.nfs* program and the cache back-end daemon *cachefilesd*.

To begin, you can download the kernel patches for kernel version 2.6.19 [4], then unpack all the patches in */usr/src/fscache* and apply them to the kernel:

```
cd /usr/src/linux
for p in $(ls ..↩
/fscache/patchset/*diff);↩
do patch -p1 < $p; done
```

Before you go on to build and install the kernel and reboot your system, it is important that you configure the kernel as shown in Figures 2 and 3 to tell NFS to use the cache.

Another Unionfs: AuFS

AuFS was inspired by Unionfs and offers the same basic functionality: it merges multiple directories (branches) to a new filesystem. However, AuFS uses its own design and is a completely new implementation. This makes AuFS simpler, more secure, and faster. Other important advantages include:

- *mmap* support
- *loopback* mounts as branches
- efficient support for sparse files
- manipulation of files in a branch

AuFS also has a Unionfs-compatible tool, *unionctl*, that supports dynamic modification of a unified filesystem: the tool is only available in compatibility mode. The typical approach in AuFS is to use the *mount* command with special options (see the man page). Of course, some restrictions exist, but most of these apply to Unionfs, too. The most important restrictions are:

- no NFS export
- no SMP support

Various tests show that AuFS is probably the better Unionfs right now.

Listing 1: /etc/cachefilesd.conf

```
dir /var/fscache # ext3,fsc
   cache-filesystem
tag mycache   # Cache name
# Block borders
brun 10%     # above normal operations
bcull 7%     # clean up cache
bstop 3%     # below cache stop
# Inode borders
frun 10%     # above normal operations
fcull 7%     # clean up cache

fstop 3%     # below cache stop
```

Once the kernel build is in progress, you can install the *cachefilesd* [4] by typing *make install*. This should be no problem, assuming your system has *glibc* version 2.4 or newer. The last preparatory step creates a new *mount.nfs* mount helper, which understands the new *fsc* option to announce the cache. An RPM package that contains the sources for the NFS tools and the FS-Cache project patches is available [5]. With the use of *rpm2targz*, you can convert the package and unpack. To keep this action as non-invasive as possible, call the new version *mount-fsc.nfs*:

```
cd nfs-utils
tar jxvf
nfs-utils-1.0.9.tar.bz2
for p in $(ls *patch); ➥
do patch -p0 < $p; done
cd nfs-utils-1.0.9
./configure ➥
--disable-nfsv4 ➥
--disable-gss
make
cd utils/mount
make
cp mount.nfs /sbin/
mount-fsc.nfs
```

As mentioned, the new mount helper announces the use of a local cache by means of the *fsc* option; the *cachefilesd* back-end daemon then enables the cache (see also Listing 1). To allow this to happen, you need an *ext3* filesystem with extended attributes enabled:

```
mount-fsc.nfs 192.168.39.1:➥
/export/test /mnt/test/N -o ➥
fsc,nolock,tcp
mount -t ext3 /dev/hdc1 ➥
/var/fscache -o ➥
user_xattr cachefilesd -s
```

Of course, you can't use this approach for a diskless client because its kernel will not support the *fsc* option for the *nfsroot* kernel parameter. Fortunately, this is not necessary, in that the initial RAM filesystem (*initramfs*) gives you a far more powerful tool.

FS-Cache in Action

The following example clearly shows the page-based approach the cache uses. The *big.tgz* file weighs in at over 100MB.

```
cd /mnt/test/N
ls -l big.tgz
-rw-r--r-- 1 1000 users
115399526 Jan 23 21:19 big.tgz
```

And the cache is almost empty:

```
df /var/fscache
Filesystem     1K-blocks
Used Available Use% Mounted on
/dev/hdc1       1031800   18200
961188  2% /var/fscache
```

The *file* command only inspects the first 256KB of a file at the most:

```
file big.tgz
big.tgz: gzip compressed data,
from Unix, last modified:
Mon Jan 1 19:03:43 2007
```

The file components are stored as a sparse file in the cache:

```
df /var/fscache
Filesystem     1K-blocks
Used Available Use% Mounted on
/dev/hdc1       1031800
18692  960696  2% /var/fscache
```

If the whole file is read, the cache grows accordingly:

```
cat big.tgz > /dev/null
df /var/fscache
Filesystem     1K-blocks
Used Available Use% Mounted on
/dev/hdc1      1031800  131020
848368 14% /var/fscache
```

Without discussing *initramfs* in more detail, we will describe the workaround for setting the kernel option. The admin must add the *mount-fsc.nfs* option, including dependent libraries, and the *busybox* framework. This enables the *init* script to mount the NFS root directory with the cache option. The script can search for a suitable cache partition on the local disk, mount the partition, *switch_root*, and launch the cache back-end daemon. This configuration gives you an ideal approach to running rich or thin clients in combination with AuFS. From an NFS root directory, you could boot an unmodified Gentoo system with FS-Cache [6] and AuFS support. Some details on how to achieve this, including a sample *initramfs*, are available [7]. ■

INFO

[1] Unionfs homepage: *http://www.fsl.cs.sunysb.edu/project-unionfs.html*

[2] AuFS homepage: *http://aufs.sourceforge.net*

[3] "FS-Cache: A Network Filesystem Caching Facility" by D. Howells *in* Ottawa Linux Symposium 2006, *http://www.linuxsymposium.org/2006/linuxsymposium_procv1.pdf*

[4] FS-Cache kernel patches: *http://people.redhat.com/~steved/fscache/nfs-utils/1.0.9-5/*

[5] FS-Cache user space patches: *http://people.redhat.com/~dhowells/fscache/patches*

[6] FS-Cache *cachefilesd*: *http://people.redhat.com/~dhowells/fscache/cachefilesd-0.8.tar.bz2*

[7] Gentoo diskless clients: *http://mozart.informatik.fh-kl.de/download/Software/GentooDiskless/gdxs.html*

THE AUTHORS

Andreas Bander is a student of Applied Computer Science at the FH Kaiserslautern in Zweibrücken, Germany, and has focused on Linux and other free operating systems for five years

Torsten Kockler is the assistant to Prof. Wilhelm Meier at the Department of Computer Science/Microsystem Technology for operating systems and programming languages at the FH Kaiserslautern.

Prof. Wilhelm Meier is a lecturer for operating systems at the Department of Computer Science/Microsystem Technology, FH Kaiserslautern.